BUILDING MULTI-AGENT RESEARCH TOOL (MART) USING AGENT ORIENTED SOFTWARE ENGINEERING (AOSE) AND AGENT TOOL

by

MADHUKAR KUMAR

B.A., (Hons), University of Delhi, India, 1995

M.A., University of Delhi, India, 1999

M.S., Kansas State University, 2003

A REPORT

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SOFTWARE ENGINEERING

Department of Computing and Information Sciences
College of Engineering

Kansas State University

Manhattan, Kansas

2003

Approved by

Major Professor
Dr Scott A DeLoach

ABSTRACT

Multi-Agent Research Tool (MART) is a multi-agent system that helps a researcher/author in writing an article by automatically searching and dynamically generating links while an article is being typed in Microsoft Word. The project, built over a period of approximately ten months, is implemented in Java. agentTool was used to develop the analysis and design of the system. The critical properties of the system were modeled in USE and OCL. The system consists of several fairly independent agents that can be placed and executed on any machine on a local network. The main agent is triggered when the user clicks a button from MS Word, which in turn finds other agents, establishes connection with them, requests them to do a search with given keywords and then collects the result and makes an html page on the user's computer. When the user clicks on another button in MS Word, the final results page is displayed in Internet Explorer.

TABLE OF CONTENTS

TABLE OF CONTENTS ii

LIST OF FIGURES iii

CHAPTER 1 – VISION DOCUMENT 1

CHAPTER 2 – PROJECT PLAN 12

CHAPTER 3 – ARCHITECTURE DESIGN 21

CHAPTER 4 – COMPONENT DESIGN 44

CHAPTER 5 – SOFTWARE QUALITY ASSURANCE 89

CHAPTER 6 – TEST PLAN 93

CHAPTER 7 – ASSESSMENT EVALUATION 97

CHAPTER 8 – USER'S MANUAL 103

CHAPTER 9 – PROJECT EVALUATION 115

REFERENCES 122

LIST OF FIGURES

Figure 1 – Overview of MART ---------- 5

Figure 2 – Overview of MART structure ---------- 7

Figure 3 – Critical Use Case ---------- 9

Figure 4 – Gannt Chart ---------- 14

Figure 5- Goals of the system ---------- 21

Figure 6 – ReadUserPreferences ---------- 23

Figure 7 – Search The Web ---------- 24

Figure 8 – Generate Results ---------- 25

Figure 9 – Role Diagram ---------- 26

Figure 10 – Agent Template Diagram ---------- 27

Figure 11- Deployment Diagram ---------- 28

Figure 12 – Class Diagram after transformation (with components) ---------- 29

Figure 13 – Object diagram ---------- 30

Figure 14 – Component Diagram of Entire System ---------- 31

Figure 15 - KeyObjectPackage ---------- 32

Figure 16 - AgentKeyPackage ---------- 33

Figure 17 – GooglePackage ---------- 34

Figure 18 - TeomaPackage ---------- 35

Figure 19 – ProducerPackage ---------- 36

Figure 20 – ManagerPackage ---------- 38

Figure 21 – USE Model ---------- 40

Figure 22 – Overview of System ---------- 44

Figure 23 – KeyObjectPackage ---------- 45

Figure 24 – Class diagram of GooglePackage ---------- 48

Figure 25 – Sequence diagram for GooglePackage ---------- 49

Figure 26 – TeomaPackage ---------- 57

Figure 27 – Sequence Diagram for TeomaPackage ---------- 58

Figure 28 – AgentKeyPackage ---------- 65

Figure 29 – ProducerPackage ---------- 72

Figure 30 – AgentProducer Sequence Diagram ---------- 74

Figure 31 – ManagerPackage ---------- 79

Figure 32 – ManagerPackage Sequence Diagram ---------- 80

Figure 33 – User Interface ---------- 103

Figure 34 – Conducting Search Without Preference ---------- 105

Figure 35 – Conducting Search With Preferences ---------- 107

Figure 36 – Results Page After Clicking DisplayResult Button ---------- 108

Figure 37 – Installing Macros 1 ---------- 110

Figure 38 - Installing Macros 2 ---------- 110

Figure 39 - Installing Macros 3 ---------- 111

Figure 40 - Installing Macros 4 ---------- 112

Figure 41 - Installing Macros 5 ---------- 113

Figure 42 – Phase 1 Break down ---------- 119

Figure 43 – Phase 2 Breakdown ---------- 119

Figure 44 – Phase 3 Breakdown ---------- 120

CHAPTER 1 – VISION DOCUMENT

Introduction

Motivation

Writing articles is an important part of work for a researcher at a university or a content provider working for a media company. While writing research or news articles, the author often conducts searches on the World Wide Web (WWW) and dig out relevant information that can be used to write the article. However, when an author is writing an article, it is often a distraction to stop writing, visit a few search engines, conduct keyword searches, retrieve relevant information and then use them in the article. This is not a very efficient especially when the author in writing an article under a deadline.

The motivation of the Multi-Agent Research Tool (MART) is to develop a research tool that helps the author to research while writing an article and save valuable time. This means that the research tool should not only be smart and efficient in conducting searches, it should also be able to work in the background fairly independently and at the same time be non-intrusive to the user. Moreover, since the Internet has now become a very common commodity and it is assumed that a person using MART has access to the Internet, it would be further more useful if the research tool could use distributed computing to retrieve research material and present them to the user whenever he/she decides to view or use them.

MART is intended to be such a tool.

Multi-Agent Systems

Multi-Agent Systems are one of the recent programming paradigms used to build intelligent software agents. An intelligent agent can be defined as some object that is fairly autonomous, goal driven, pro-active and reactive. This means that intelligent agents should be able to "inhabit" an environment and not only should they be capable of

responding to external stimuli, they should be fairly independent and have an end result in their "mind", while going about doing small set of activities.

There are several methodologies for building Multi-Agent systems such as Gaia, Societies in Open and Distributed Agent (SODA), Methodology for Engineering Systems of Software Agents (MESSAGE) and Multi-Agent Software Engineering (MaSE).

MART will be built in the multi-agent programming paradigm and it will use MaSE as the Agent Oriented System Engineering (AOSE) methodology to construct the system.

Multi-Agent Software Engineering (MaSE)

Of all the Agent Oriented Systems Engineering (AOSE) methodologies, probably the simplest, the most logical and the most elaborate is the Multi-agent Systems Engineering (MaSE) methodology.

Like most of the other methodologies, the MaSE methodology also has an analysis phase and a design phase. The analysis phase yields goals, use cases and roles of the overall system while the design phase results in the development of agent classes, tasks and conversations. To build an agent's system under the MaSE methodology, a developer must use the agentTool that not only helps in designing the system but also generates the stub code of the entire system in Java.

Under MaSE, the first step in the analysis phase is to break down the functional requirements of the overall system into a set of hierarchical goals. These goals are primarily of four types – summary, partitioned, combined and non-functional. Once the goals of the system have been identified, the next step in the analysis phase is to apply use cases and to develop sequence diagrams. At this state the aim is to develop as many event sequences of the system as possible without repeating any events.

The last step in the analysis phase is to develop roles and refine them from the

goals and the sequence diagrams of the first two steps. Usually, the mapping of goals to roles is one-to-one, in the sense that one goal maps to one role. However, closely related goals may be combined into a single role. It is ultimately the overall system requirements and its characteristics that decide the goals, use cases and roles. After the roles are created, tasks associated with each role that describe the role's behavior and that are required to achieve the goals associated are developed. The role diagram is the first step in MaSE which defines the interactions between different tasks of different roles. Here the tasks are also developed in the form of state diagrams. These tasks are either transient or persistent. These individual tasks are then developed in to their concurrent task models which have an initial state, a termination state and transitions between the two and/or other states.

The design phase in MaSE starts after all the roles and tasks have been defined. Once this is done the first step in the design phase is now to develop agents by assigning them roles. Since the roles reflect the overall goals of the system and can be manipulated modularly, the authors refer to them as a bridge between the analysis and the design phase of this methodology. It is up to the system designer on how many agents finally show up in the overall system and what roles each agents play. Once the agents have been defined, the designer should now connect the agents using conversations. Each conversation is designed from both the initiator as well as the responder's perspective. These conversations are much like the state diagrams that have initial and end states. A designer can also develop other states inside the responder and the initiator keeping in mind the previous tasks and roles and finally verify to see whether the conversations are deadlock free or not. The verification is done by the Spin tool which is embedded inside the agentTool.

Another advantage of using MaSE is that while designing the individual agents, it

is totally up to the designer on what architecture to use for making the internal components of the agents. It is here that Beliefs, Desires and Intentions (BDI) architecture can be used to design the agent architecture or the designer can choose his/her own architecture and make the components accordingly.

Finally the structure of an instance of the entire system can be developed using the deployment model in which several systems are designed, the agents are placed in the systems from the list of agents and the agents are made to talk using conversations already developed from the previous step.

Project Overview

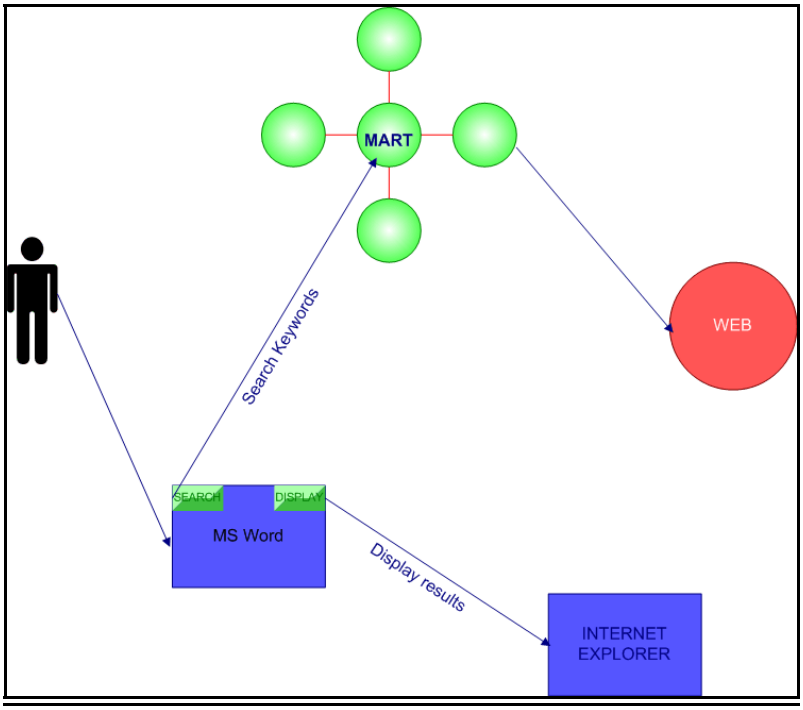


Figure 1 - Overview of MART

The MSE project involves building the Multi-Agent Research Tool (MART) software that will help researchers and authors to dynamically search for information and write articles. Put simply, MART is intended to be a software package that will be integrated with one of the most commonly used word processor program, MS Word ® and help the user to conduct relevant searches in the background based on the words that have been typed in the article and then display the results to the user.

The primary goal of creating MART is develop a smart Multi-agent system that would intuitively and independently search and display information about an article while it is being typed by a researcher.

The main feature of the software is to do relevant searches and display the results dynamically even while the user is typing the article and the real challenge is to find

results that are relevant to the article being written and are not redundant.

Goal

To develop a tool that will be integrated with MS Word and help an author in searching research information for an article based on the words that have already been typed in an article. The search results should be fairly accurate, relevant to the current article being typed and non-redundant.

Purpose

To build a multi-agent system with a collection of intelligent software agents that are fairly autonomous and use distributed computing to achieve a common goal.

Features

1. Integrated with MS Word.
2. There will be separate agents responsible for specific tasks. For example, one agent should be a specialist only in conducting searches in Google.
3. The search should be conducted only when the user want to use MART.
4. The user should be able type in the preferences for the search results. For example, the user should be able to specify that the results should be only from a specified website and should only between two certain dates.
5. Allow the user to choose when he/she wants to view the results.

Requirement Specification

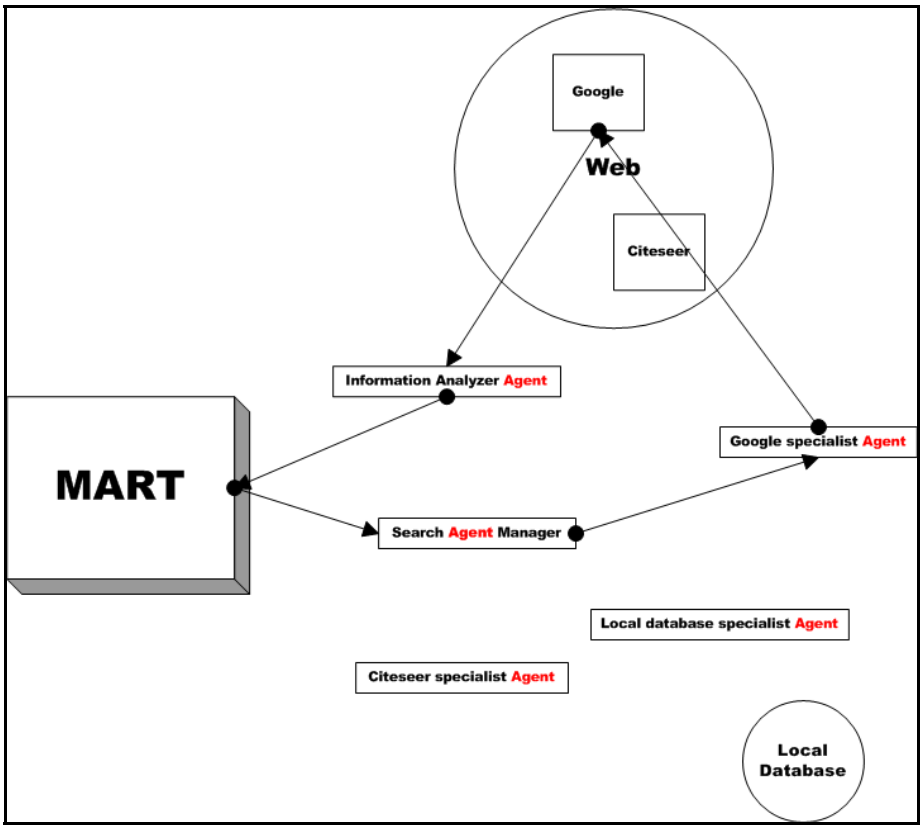


Figure 2 - Overview of the MART structure

Main requirements

1. **Multi-Agent Systems based** architecture. Specific functions will be performed by individual agents that are fairly independent, goal driven, proactive and reactive.

2. Use of **Multi-Agent Software Engineering (MaSE) methodology and agentTool** in developing the system.

3. The final product, MART should be integrated with MS Word ® processor.

4. The individual agents should be able to **run on any machine** that has a Java virtual machine. This includes machines running on the following platforms – Windows, Linux, Macs.

5. Internet Explorer ®, a common web browser used by Windows users, will be used

to display the search results.

6. For testing and demonstration purposes, the software should use the **Google API** to conduct searches on Google search engine.

7. Failure of one agent should not crash the whole software.

Specific Requirements

1. Allow a user to write an article in MS Word ®.

2. The user interface would include buttons embedded inside MS Word ® which will allow the user to search, save and display the results.

3. The program will generate HTML 4.0 result pages so that they can be viewed using a common web browser - Internet Explorer ®.

Critical Use Cases

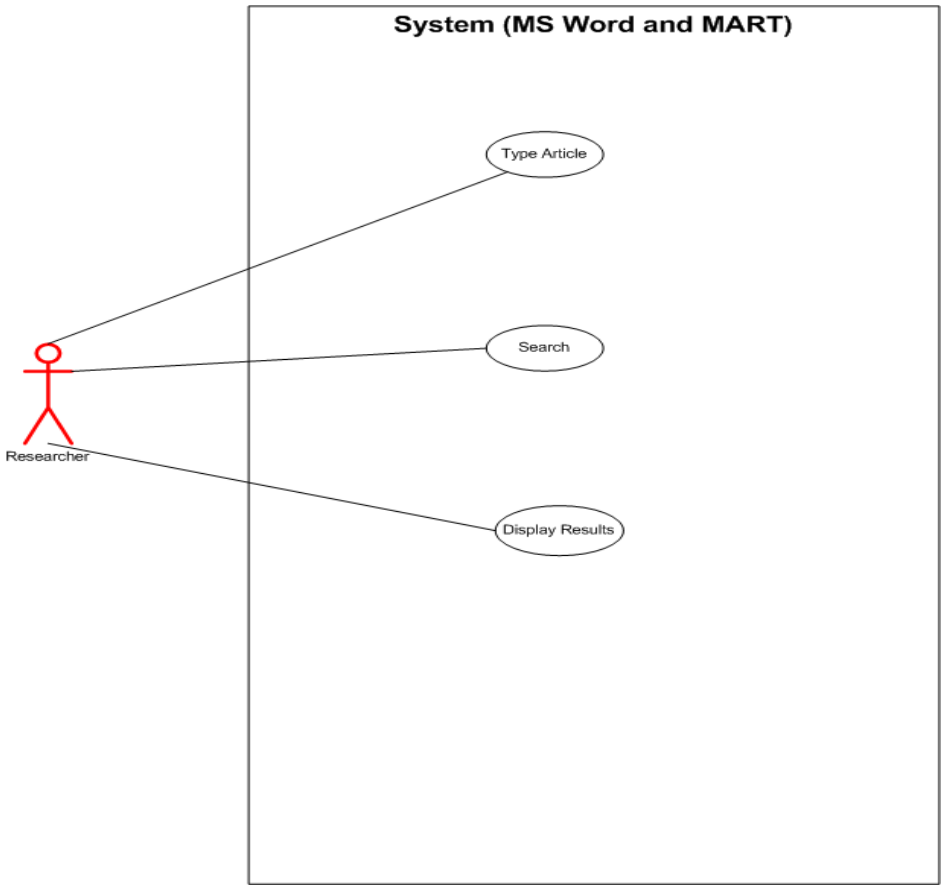


Figure 3 - Critical Use Case

<u>Use Case 1:</u> Type article in MS Word

Description - The user should be able to type an article in MS Word with his/her normal settings and preferences and when MART is finally integrated with the word processor none of the features, preferences and settings should either change or stop working. The user should also be able to save the articles as any other regular files.

<u>Use Case 2:</u> Search for research material

Description – The user should be able to click on a button somewhere in the MS Word menu that should trigger off the MART search agents. Once the button is pressed, the agents should collect all the words from the article being currently typed out in the

article, figure out the relevant keywords, conduct a search with those keywords using some search engines and then create an HTML file with all the valid results.

Filter results: The user should be able to specify that the results should only be from a particular web site or that the results should be filtered by date. The user should also be able to specify that only certain kinds of files (like doc, html or pdf) files should be searched.

Keywords ranking: The program should be able to rank the words typed by the user using some algorithm.

Search constraint: Some of the words typed by the user should be at least present in the final results page.

Use Case 3: Display research material

Description – The user should be able to click on a button somewhere in the MS Word menu that should immediately open a window of Internet Explorer ® browser that will have all links to web sites with information relevant to the current article being typed in MS Word.

Assumptions

1. The user has an internet connection when ever he/she is using MART.
2. Every agent knows the port numbers of the every other agent that they need to collaborate with.

Constraints

1. Since the Agent Tool generates stub code in Java, the program will be developed in Java.
2. The Google API only returns the first 10 search results per query.
3. There is a limit to the number of searches a program can conduct every day using the Google API.

Environment

1. The software package will be compiled using the JDK 1.4.0

2. agentTool will be used for the design and analysis part of building the system and to generate the stub code.

3. IBM Visual Age will be used for coding the software package.

4. USE 2.0 along with Object Constraint Language (OCL) will be used for formal specifications.

5. The software package will be tested on Windows XP and the individual agents will be tested on Windows XP, Red Hat Linux 9.0 and Unix Solaris 2.7.

CHAPTER 2 – PROJECT PLAN

Individual tasks breakdown

Inception phase

The inception phase would involve development of a prototype that would display the feasibility of the project and also give an idea to the users about how the final product will finally look and behave.

The phase will also involve the development of documentations that will be presented during the first presentation. These documents would include a vision document that will outline the requirements and the functionalities of the product, a project plan and a Software Quality Assurance (SQA) plan.

The requirements phase will be over once the committee approves the documents and the executable prototype after the first presentation, with or without changes.

Elaboration Phase

The inception phase would involve developing the overall architecture of the product using the agentTool. The agentTool program will be used to develop the overall goals of the system (MART), the roles and the concurrent tasks.

Specific use cases inside the system will also be designed in this phase using the Agent Tool. Finally, an agent template will be developed that will outline the different agents and the way they will communicate with each other through conversations. The conversations will also be verified using the Spin tool. A Deployment plan, also developed inside the Agent Tool, will also be developed in this phase.

The design phase will also involve development of a second executable prototype that will be demonstrated during the second presentation in front of the committee members. In the elaboration phase the vision document, the project plan and the software quality assurance plan document will also be revised based on the feedback provided by

the committee members during the first presentation.

The design phase will be over once the committee members approve the second prototype and the design artifacts developed during this phase with or without changes.

Production Phase

The production phase would involve generation of stub code from the agentTool design file (MAML file) and developing all the individual agents that would perform separate tasks. This phase would also involve development of the user interface components and the search algorithm that will be incorporated in one of the agents.

Finally, this phase would involve integration of all the agents and the individual components that will constitute the whole system.

Testing

The testing phase would involve testing of all the requirements and the functionalities outlined in the vision document. This phase would also include integration testing for the entire system.

Documentation

The documentation phase would involve developing several artifacts that will submitted along with the final software at the end of the third presentation.

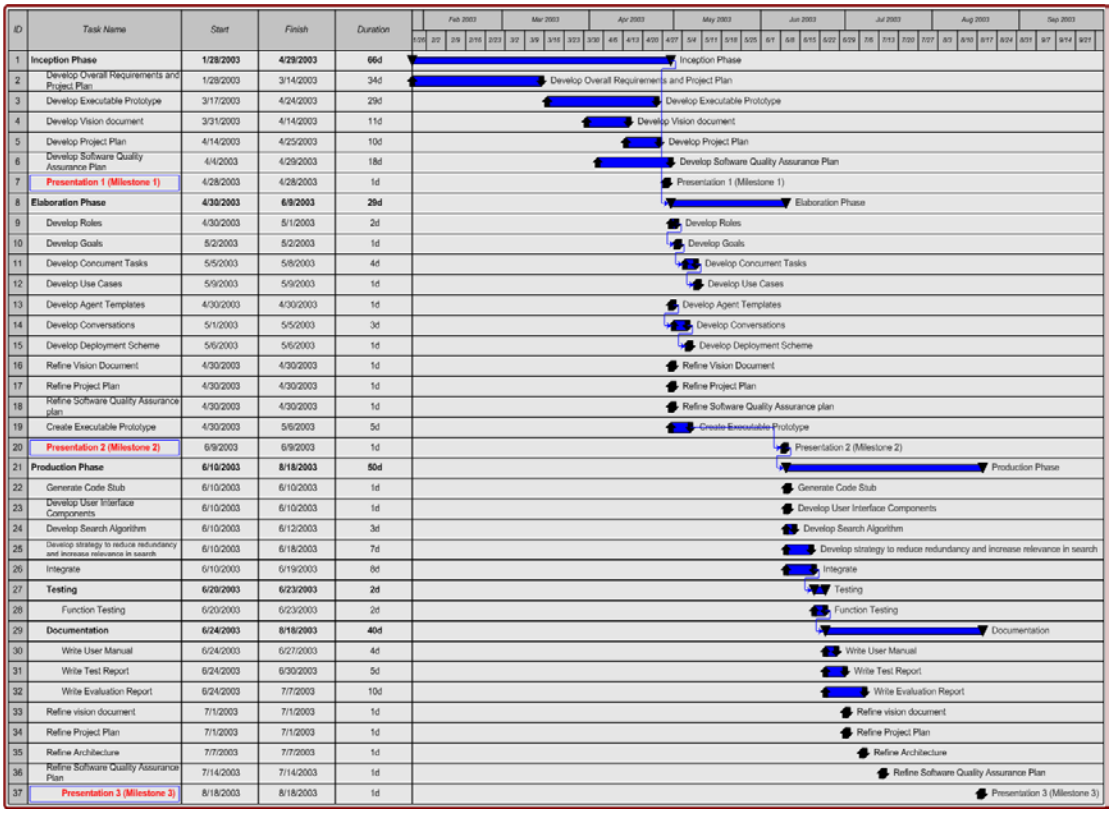
The documents would include the API web pages that would outline each and every java package with information about individual classes, attributes and methods.

The phase would also involve development of a user manual that would describe how to install and run the final software. A test report will also be developed that would outline how the tests were conducted and what were the results from the tests.

An evaluation report will also be written during this phase that would give a brief evaluation of the entire project.

The phase will be considered over once the committee members approve the final

version of the software, demonstrated during the third and the last presentation, and the set of documentation mentioned above, with or without changes.



ID	Task Name	Start	Finish	Duration
1	**Inception Phase**	**1/28/2003**	**4/29/2003**	**66d**
2	Develop Overall Requirements and Project Plan	1/28/2003	3/14/2003	34d
3	Develop Executable Prototype	3/17/2003	4/24/2003	29d
4	Develop Vision document	3/31/2003	4/14/2003	11d
5	Develop Project Plan	4/14/2003	4/25/2003	10d
6	Develop Software Quality Assurance Plan	4/4/2003	4/29/2003	18d
7	**Presentation 1 (Milestone 1)**	4/28/2003	4/28/2003	1d
8	**Elaboration Phase**	**4/30/2003**	**6/9/2003**	**29d**
9	Develop Roles	4/30/2003	5/1/2003	2d
10	Develop Goals	5/2/2003	5/2/2003	1d
11	Develop Concurrent Tasks	5/5/2003	5/8/2003	4d
12	Develop Use Cases	5/9/2003	5/9/2003	1d
13	Develop Agent Templates	4/30/2003	4/30/2003	1d
14	Develop Conversations	5/1/2003	5/5/2003	3d
15	Develop Deployment Scheme	5/6/2003	5/6/2003	1d
16	Refine Vision Document	4/30/2003	4/30/2003	1d
17	Refine Project Plan	4/30/2003	4/30/2003	1d
18	Refine Software Quality Assurance plan	4/30/2003	4/30/2003	1d
19	Create Executable Prototype	4/30/2003	5/6/2003	5d
20	**Presentation 2 (Milestone 2)**	6/9/2003	6/9/2003	1d
21	**Production Phase**	**6/10/2003**	**8/18/2003**	**50d**
22	Generate Code Stub	6/10/2003	6/10/2003	1d
23	Develop User Interface Components	6/10/2003	6/10/2003	1d
24	Develop Search Algorithm	6/10/2003	6/12/2003	3d
25	Develop strategy to reduce redundancy and increase relevance in search	6/10/2003	6/18/2003	7d
26	Integrate	6/10/2003	6/19/2003	8d
27	**Testing**	**6/20/2003**	**6/23/2003**	**2d**
28	Function Testing	6/20/2003	6/23/2003	2d
29	**Documentation**	**6/24/2003**	**8/18/2003**	**40d**
30	Write User Manual	6/24/2003	6/27/2003	4d
31	Write Test Report	6/24/2003	6/30/2003	5d
32	Write Evaluation Report	6/24/2003	7/7/2003	10d
33	Refine vision document	7/1/2003	7/1/2003	1d
34	Refine Project Plan	7/1/2003	7/1/2003	1d
35	Refine Architecture	7/7/2003	7/7/2003	1d
36	Refine Software Quality Assurance Plan	7/14/2003	7/14/2003	1d
37	**Presentation 3 (Milestone 3)**	8/18/2003	8/18/2003	1d

Figure 4 - Gannt Chart

Cost Estimate

Function point analysis

Type	Simple	Average	Complex	Total
Inputs			1. Entire Word file being typed by user	6
Outputs	1. Comments 2. Error Messages	1. Search results in an HTML Page		13
Inquiries			1. Search Keywords	6
Files		1. Results file (HTML) 2. Text file of relevant keywords		20
Interfaces	1. Microsoft Word 2. Internet Explorer			10
Total				**55**

COCOMO

Since COCOMO II is used mostly for large development teams, Constructive Cost Model (COCOMO) originally developed by Barry Boehm, will be used to estimate total cost of the project in terms of effort and time. The estimation will be based on the assumption that the MSE project is fairly uncomplicated and flexible and hence the "organic" mode will be used in the COCOMO model.

The equations used for COCOMO model is as follows:

Effort = C1*EAF*(Size)P1

Time = C2*(Effort)P2

where:

Effort = number of person-months

C1 = constant scaling coefficient for effort

C2 = a constant scaling coefficient for schedule

P1 = an exponent that characterizes the economics of scale inherent in the process used to produce the end product.

P2 = an exponent that characterizes the inherent inertia and parallelism in managing a software development effort.

EAF = an effort adjustment factor that characterizes the domain, personnel, environment, and tools used to produce the artifacts of the process.

Size = size of the end product (in human-generated source code), measured by the number of delivered source instructions.

Time = total number of months.

Since we are assuming that the current project is in the Organic mode, the following values will be used for the above mentioned equations.

C1 = 3.2

C2 = 2.5

P1 = 1.05

P2 = 0.38

Since the project has only one developer, EAF is not applicable in this situation. In the second phase once the code was generated from the design model and after the second executable prototype was built it was found that there are 17 classes instead of 9 as originally envisaged. Since each class has approximately about 130 lines of code, the total lines of code will be somewhere around 2550 instead

of 1170 as originally estimated. By plugging these numbers in the above two equations, the following results are obtained:

Effort = 3.2*(2.500)1.05 = 8.4 person-months (earlier it was 4.9 months)

Time = 2.5*(8.4)0.38 = 7.9 months (earlier it was 4.6 months)

Productivity = 2500/7.9 = 316 LOC-month (earlier it was 330 LOC-month)

Staff = 8.4/7.9 = 1 person (Earlier 1 person)

This means that the developer will require approximately eight months instead of four and a half months originally estimated to implement the project if about 316 lines of code are generated each month and if the coder works five days a week for eight hours every day.

It should be noted here that the developer may or may not be able to work for eight hours a day for five days a week only on the project since he is also expected to attend classes and do other school related work. However, keeping in mind that since there are no other team members in the development team (and hence there are no overhead management costs in communication and coordination) and the fact that the production phase is scheduled over (see Gannt Chart) the summer break, the production may actually just finish in roughly eight months. This means that the original date of finishing the project August 2003, will now move to sometime in the end of November, 2003.

Architecture elaboration plan

The following tasks should be completed before the second presentation –

Revision of Vision Document – After the first presentation, based on the suggestions provided by the committee member, the vision document will be modified. The revised document will be approved by the major professor.

Revision of Project Plan – After the first presentation, based on the suggestions provided by the committee member, the project plan will be modified. The revised document will have an updated cost estimate and will be approved by the major professor.

Architecture Design – Development of overall goals, roles, concurrent tasks, use cases, agent templates, conversation between different agents and the verification of conversations should be completed in the agentTool. In addition to the above tasks, an object model should also be developed that would describe all the objects of the system and the important methods and attributes of each object.

Development of a second prototype – Development of the second prototype should establish the fact that agents could talk to each other and return with some relevant search details if some keywords are passed as search parameters.

Test plan – Development of a test plan should show that all requirements specified in the vision document are satisfied.

Formal Technical Inspection – The architecture design shall undergo formal technical inspection by two MSE students who have agreed to participate in the inspection - Cem Oguzhan and Esteban Guillen.

Implementation Plan

The following tasks should be completed by the final presentation

User Documentation

The document will describe all the features of MART and also explain to the user how to install and use the software. The document will be considered complete when all the features have been described adequately in the document.

Revision of Architecture Design document

The revised Architecture Design document will record any changes made to the design and will also have detailed design description of the individual components of the software.

Assessment Evaluation

The Assessment Evaluation document will describe all the tests that were done on the MART software and the results of the test.

Project Evaluation

The Project Evaluation document will give an overall evaluation of the project regarding the efficiency of the methodologies used and the usefulness of reviews.

Implementing individual packages and classes

Implement AgentManager

The implementation of the AgentManager would be considered complete once it is able to perform the following tasks.

The AgentManager should be able to send out a message to the AgentKey to read the user preferences and keywords. Once the keywords are read and ranked, the AgentManager should send out messages to the Sleuth agents and get back the raw result. Finally, the AgentManager should be able to send the raw results to the AgentProducer and get back a final result file.

Implement AgentGoogle

This task will be considered complete when the AgentGoogle is be able to conduct a search on the Google web site using the keywords provided to it and then return an object with the raw results to the AgentManager.

Implement AgentTeoma

This task will be considered complete when the AgentTeoma is be able to conduct a search on the Google web site using the keywords provided to it and

then return an object with the raw results to the AgentManager.

Implement AgentProducer

The implementation of the AgentProducer will be considered complete when it is able to respond to a message by AgentManager by refining the results from raw results and sending back the final result file.

Integration

The integration will be considered complete when a user is able to click on the “Search and Save” macro that would initiate AgentManager to complete all its tasks.

CHAPTER 3 – ARCHITECTURE DESIGN

Introduction

The purpose of the document is to describe the architecture design of the project Multi-Agent Research Tool (MART) as outlined in the requirements in section 3 of the vision document. The document will outline and describe the goals, the role model, the object diagram and the sequence diagrams of MART.

Goals

The overall goal of the system is to produce searched results. This goal is partitioned into four sub goals, mainly to rank and refine the keywords that will be used for search, searching the web for results, producing the result (output file) for the user and to manage the whole distributed system

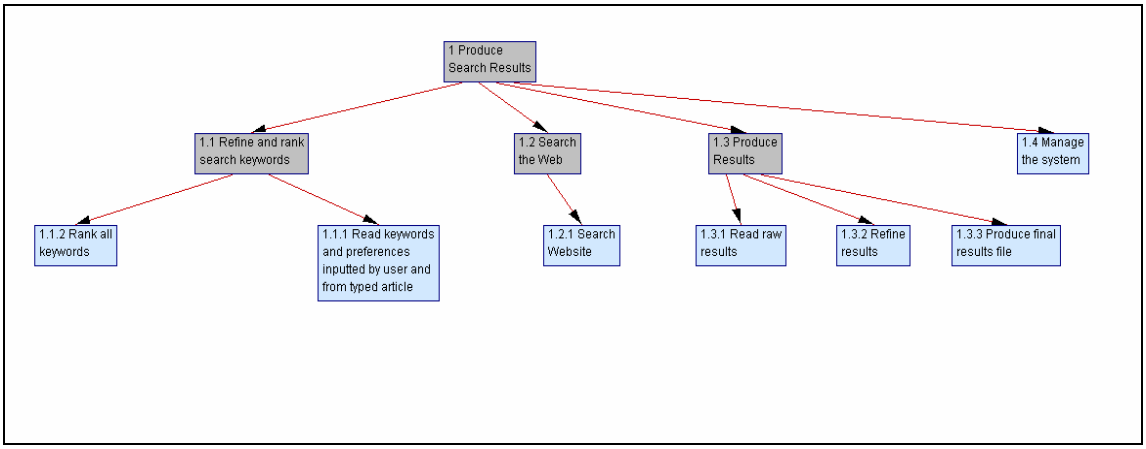

1 Produce Search Results
1.1 Refine and rank search keywords
1.2 Search the Web
1.3 Produce Results
1.4 Manage the system
1.1.2 Rank all keywords
1.1.1 Read keywords and preferences inputted by user and from typed article
1.2.1 Search Website
1.3.1 Read raw results
1.3.2 Refine results
1.3.3 Produce final results file

Figure 5 – Goals of the system

The "rank and refine search keywords" role is also partitioned and is divided into reading user inputted keywords and in ranking the keywords for search.

The "search" role is also partitioned and for the current version consists only of

searching websites. In future versions, another role could be added into it that would also conduct searches on database located on the host computer and/or local network.

The role “produce results” is partitioned into creating raw results, refining the results and producing the final result.

Use Cases

There are three primary use cases for the entire system namely “Read User Preferences,” “Search the Web” and “Generate Results.”

Read User Preferences

In this sequence diagram, the manager sends a message to the Keyword reader to read the user inputted keywords and preferences. The manager then takes this object and sends it to the Keyword ranker with a request to rank all the keywords. The Keywords ranker ranks the keywords based on the number of times they appear in the user typed article and returns the ranked keywords to the manager.

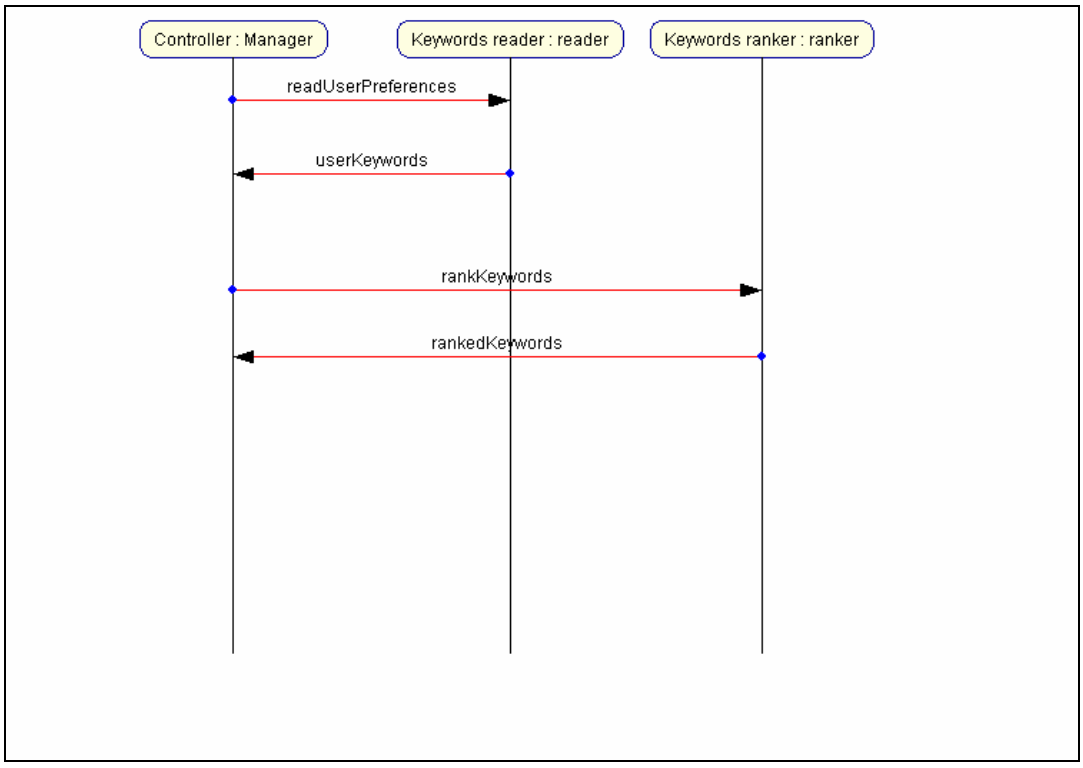


Figure 6 – Read User Preferences

Search the Web

The following sequence diagram shows that the manager sends a message to the Sleuth, along with the keywords and requests it search the web. The sleuth conducts the search and returns the raw result to the manager.

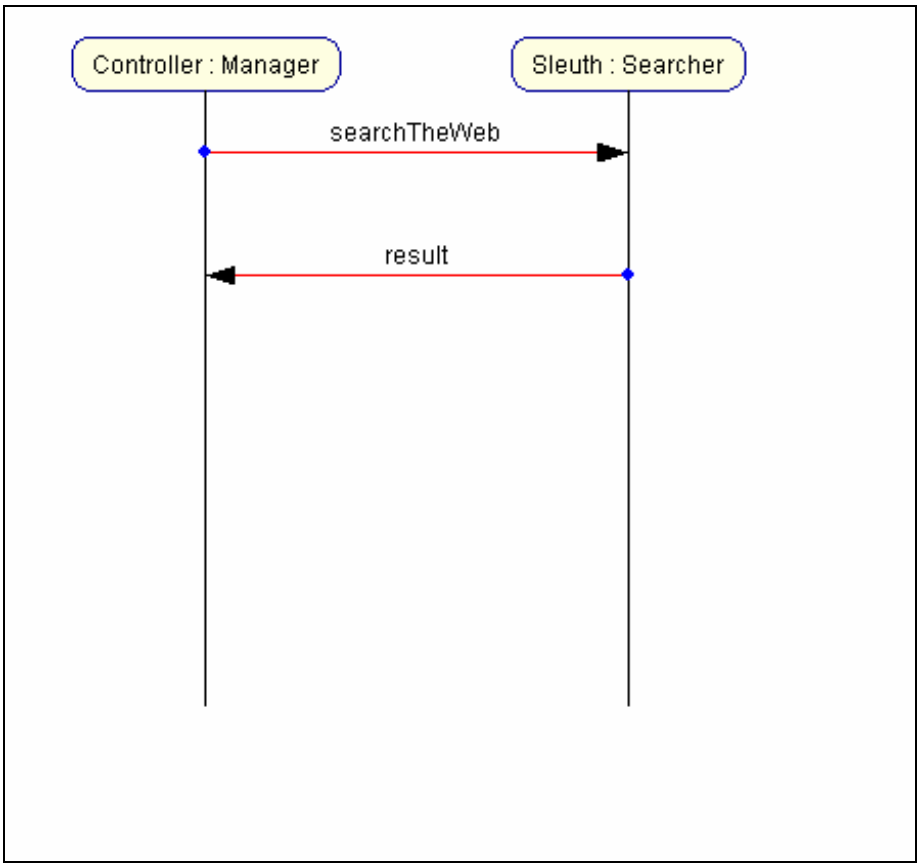


Figure 7 – Search the Web

Generate results

The following diagram show that the Manager, once it has the raw results, sends a message to the Result producer along with the raw results. The producer refines the results and then sends back the final result to the manager.

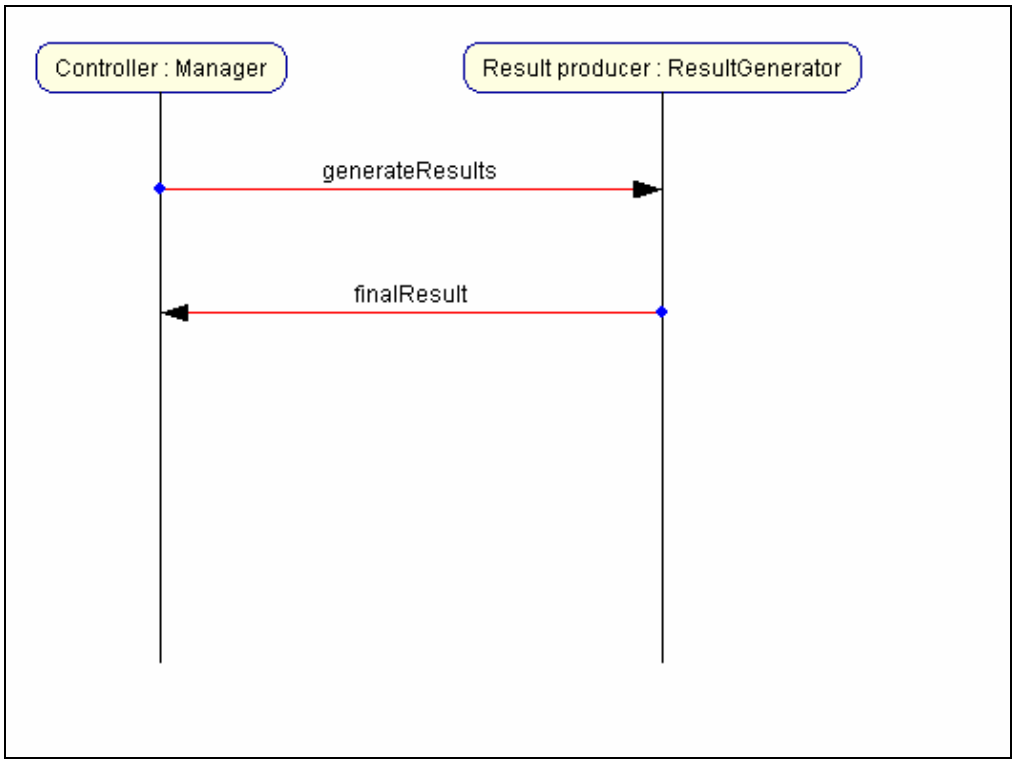


Figure 8 – Generate results

Role diagram

The role diagram depicts how the different goals are mapped to the roles of the system. The following diagram shows that the controller has a bunch of tasks that collaborate with the other roles in order to read keywords, perform a search and generate the final result.

The dotted line in the Sleuth role (makeRaw) shows that it is an internal method call. The makeRawResults task is invoked by the rawResults task of the Sleuth once it receives the searchTheWeb request from the controller.

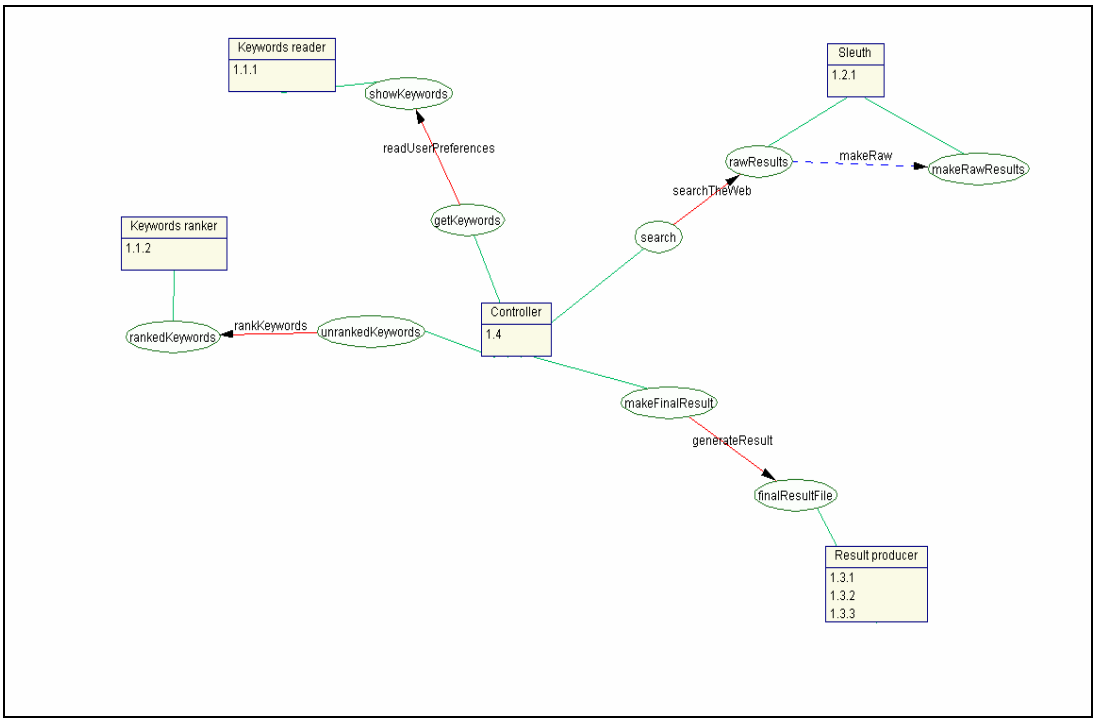


Figure 9 – Role Diagram

Agent Template diagram

The agent template diagram shows that the current version of the software (MART) has five different agents namely AgentManager, AgentKey, AgentProducer, AgentGoogle and AgentTeoma. The red lines connecting between the agents represent the conversations that are initiated between the agents. For example searchTheWeb is a conversation that is initiated by AgentManager and it requests the sleuth agents (AgentGoogle and AgentTeoma) to conduct a search and return the raw results.

The different agents in the following diagram represent independent entities with their own separate main methods. These agents could be placed on different machines and still be able to talk to each other using the conversations defined in the system.

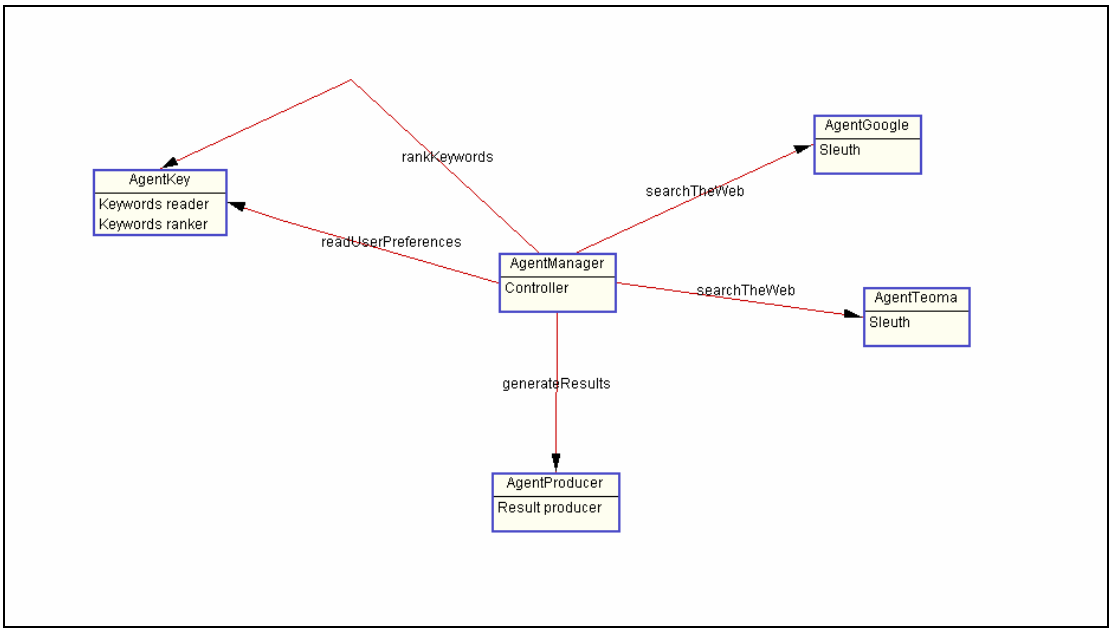


Figure 10 - Agent Template diagram

Deployment diagram

The following diagram shows that since the agents are independent entities they can be placed on different systems (machines or networks) and yet talk to each other using pre-defined conversations.

The following diagram also shows that for the current version of the software the AgentManager and AgentKey will be on the same machine as the one which has the Microsoft Word running on it. This is because both the AgentManager and the AgentKey need to have access to the file that the user is typing the article on.

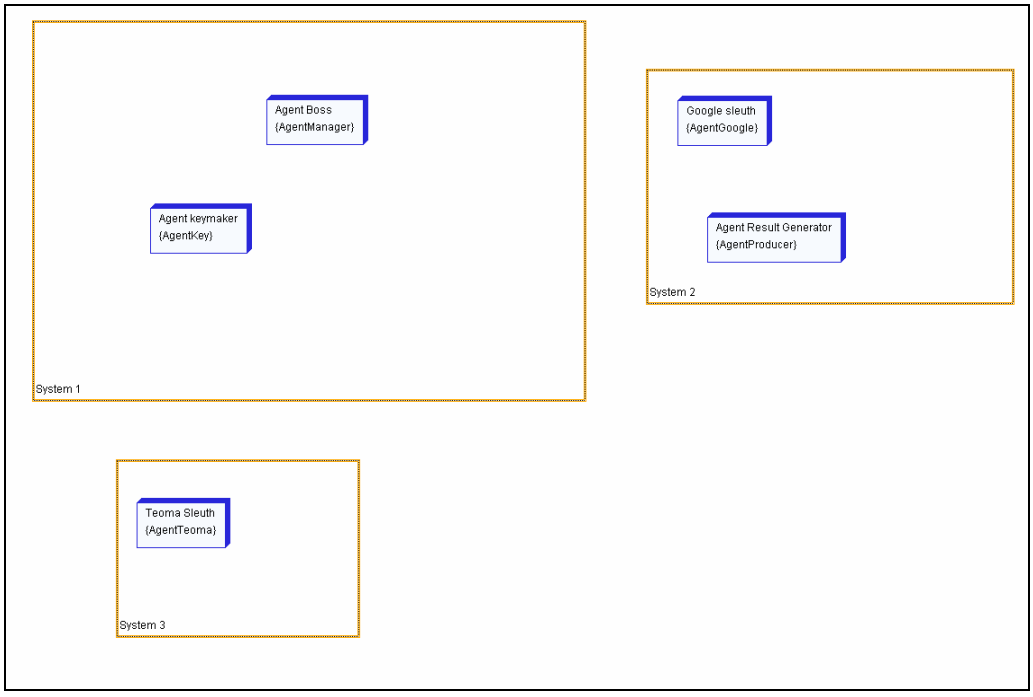


Figure 11 – Deployment diagram

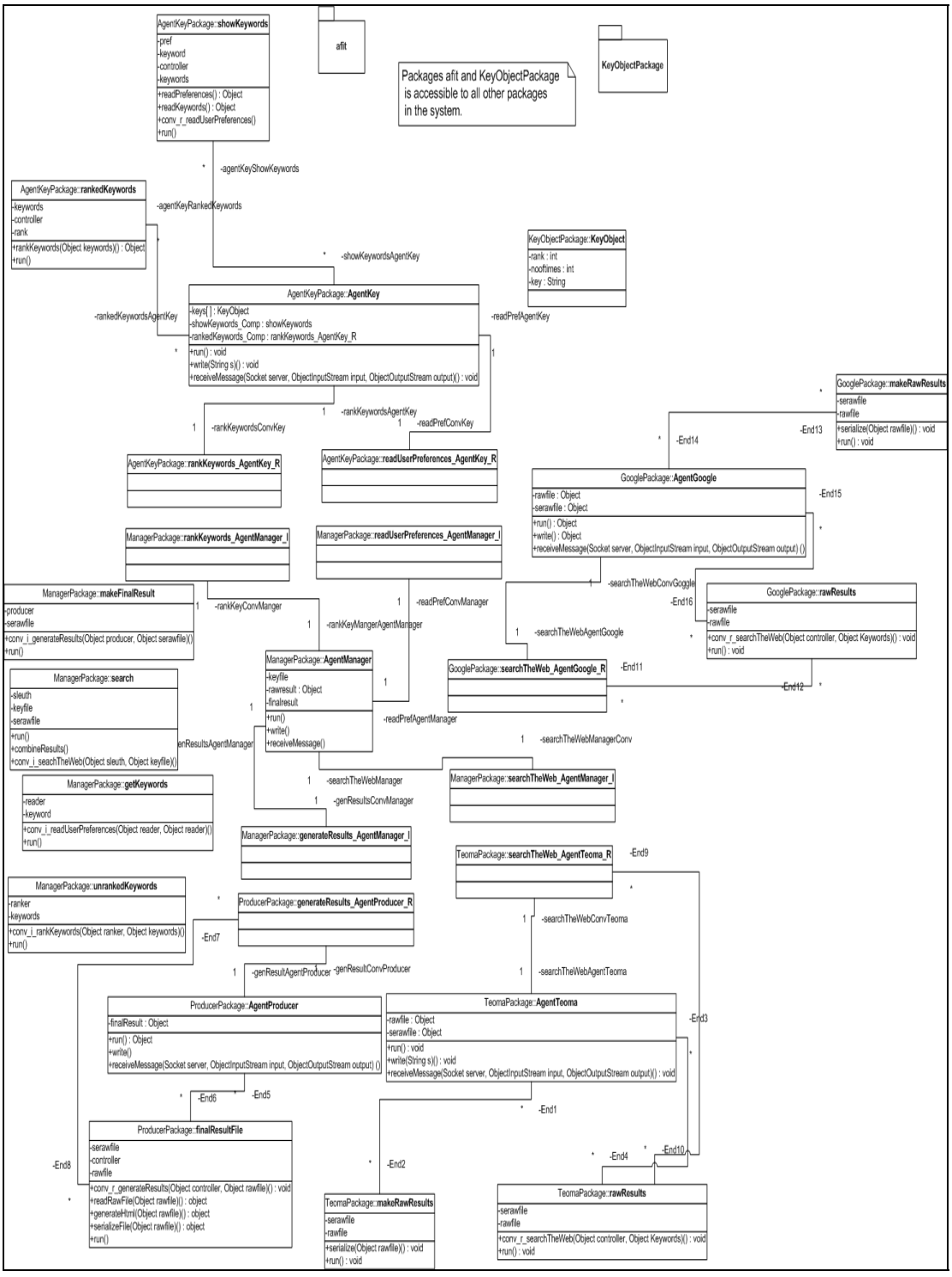


Figure 12 – Classes generated after transformation (with components)

Object Model

When AgentTool generated code using "newAgentMom", it made separate

classes for each component. The object model for the code generated is shown in Figure13. However, since it is simpler to have components as separate methods in the agents, the object model was redesigned as shown Figure 14.

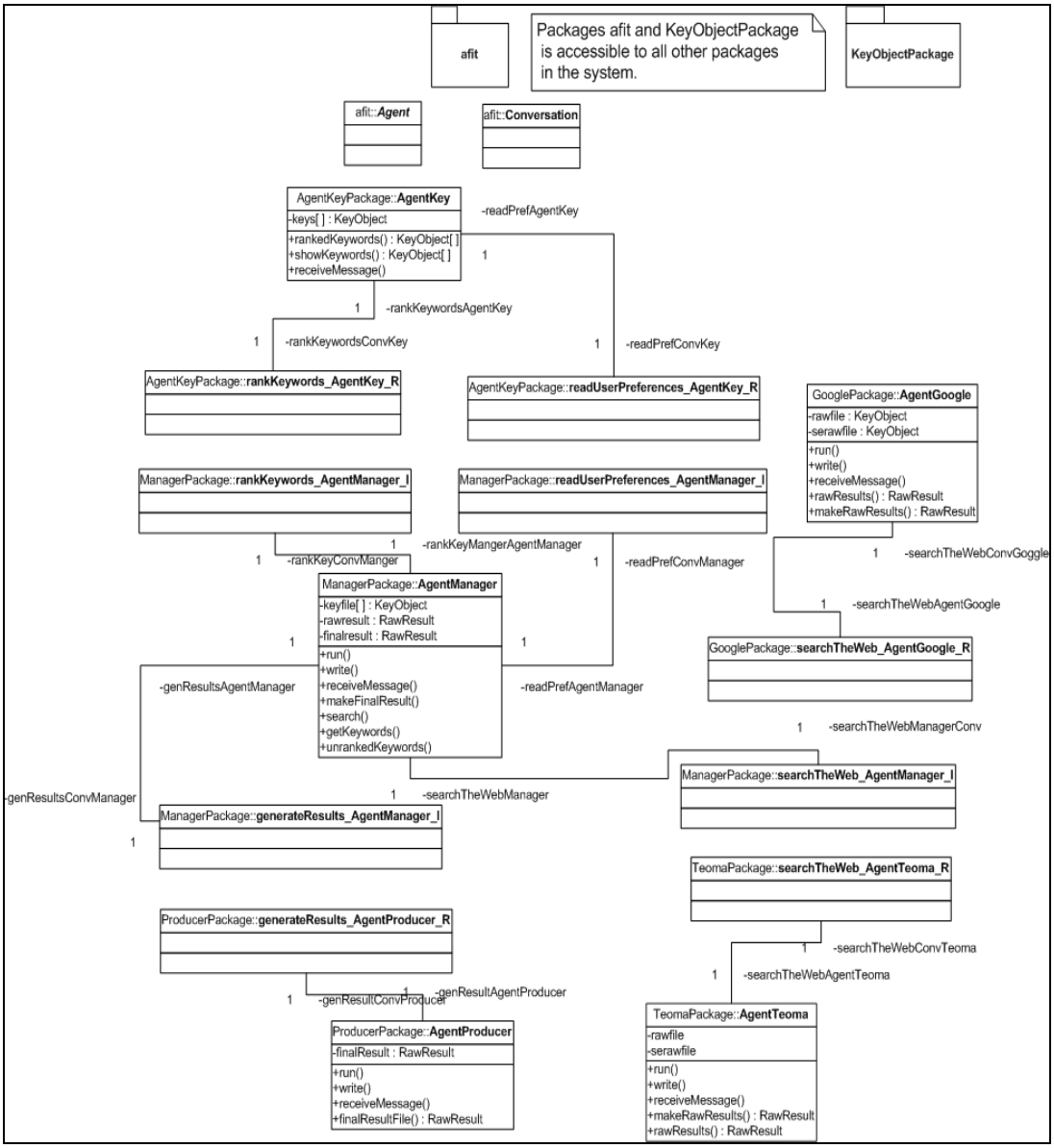


Figure 13 – Object diagram

The object model shows each and every class of the software and the associations

between them. The packages, “afit” and “KeyObjectPackage” are accessible to all the other packages in the system. The KeyObjectPackage hold the classes that store the keywords and the final result of the search. The primary packages (representing the agents and their component classes) are AgentKeyPackage, ManagerPackage, GooglePackage, TeomaPackage and ProducerPackage. Figure 15 shows the interaction between all the packages. ManagerPackage holds the AgentManager class which is the main controller of the whole system and initiates all the conversations with other Agents.

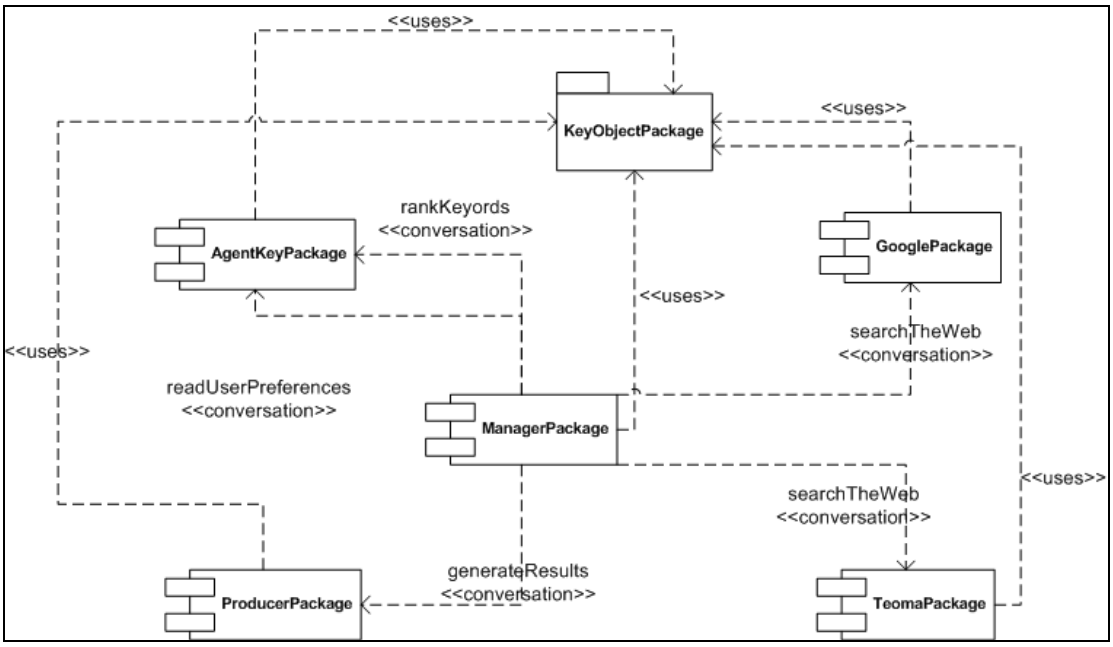


Figure 14 – Component diagram of the entire system

KeyObjectPackage

The KeyObjectPackage is the data package that is used by all other packages to send results and keywords to each other. Class KeyObject is a simple class with three main attributes called name, rank and nooftime. The actual keyword is stored in the variable name while the integers rank and nooftime record the ranking of the keyword as compared to the other keywords and the frequency with which it appears in the typed article respectively.

The RawResult class is also a simple object that holds XML(html) string in its main attribute called “result.”

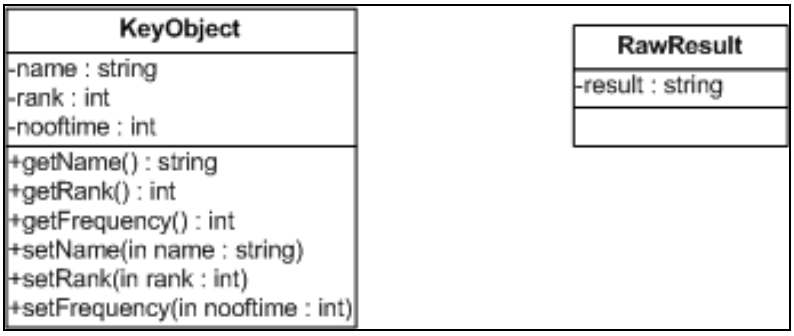


Figure 15 – The KeyObjectPackage

AgentKeyPackage

AgentKeyPackage has three classes, the AgentKey, rankKeywords_AgentKey_R and readUserPreferences_AgentKey_R. The AgentKey is the main class of the package and has the main method as well as methods to read user keywords and rank keywords. When AgentKey receives a message from AgentManager (it does not respond to message from any other class), it creates the instances of conversation responder class based on the kind of message. The responder classes then invoke the methods that actually reads user inputted keywords and ranks them.

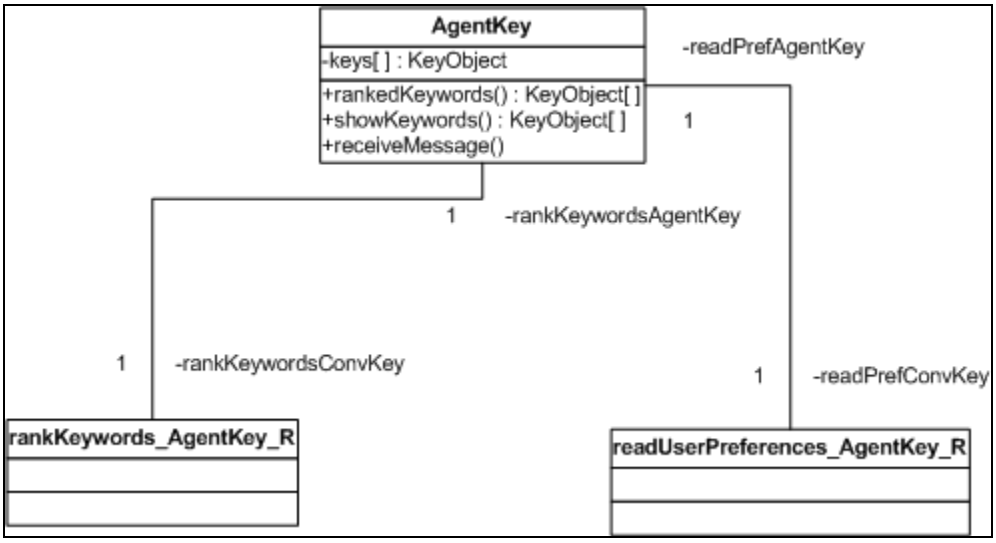


Figure 16 – AgentKey package

GooglePackage

GooglePackage has two classes AgentGoogle and searchTheWeb_AgentGoogle_R. AgentGoogle has one important method called search. When AgentGoogle receives a message from AgentManager, it creates an instance of the searchTheWeb_AgentGoogle_R that handles the conversation from there on. The conversation is closed once AgentGoogle conducts the search, serializes the result and sends back the serialized object (in the form of an instance of RawResult class in KeyObjectPackage) to the AgentManager.

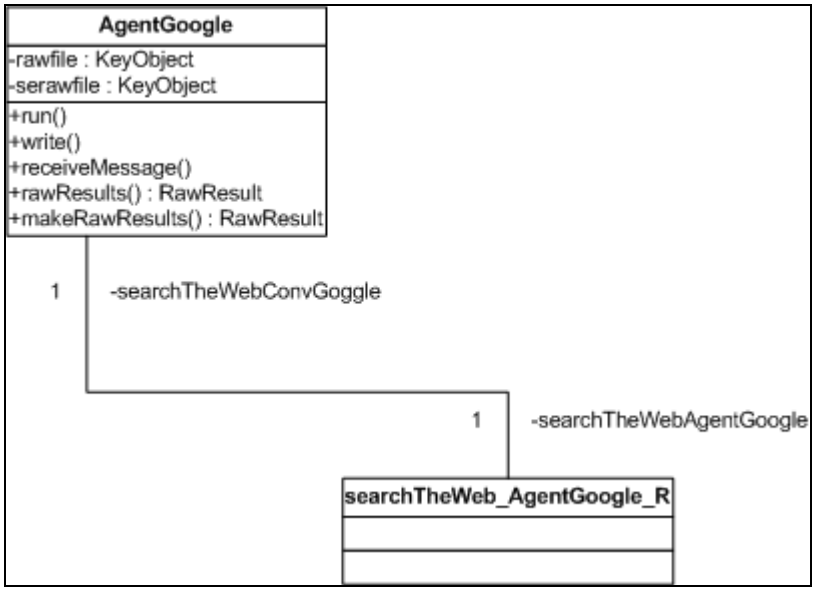


Figure 17 –GooglePackage

TeomaPackage

TeomaPackage is similar to the GooglePackage and consists of the following two classes AgentTeoma and searchTheWeb_AgentTeoma_R. These two classes perform the same functions like the classes in the GooglePackage. The main difference between the two packages lie in the way AgentGoogle and AgentTeoma conduct searches. AgentGoogle uses Google API to conduct keyword search on Google.com while AgentTeoma opens a URL connection with the website and gets back raw results.

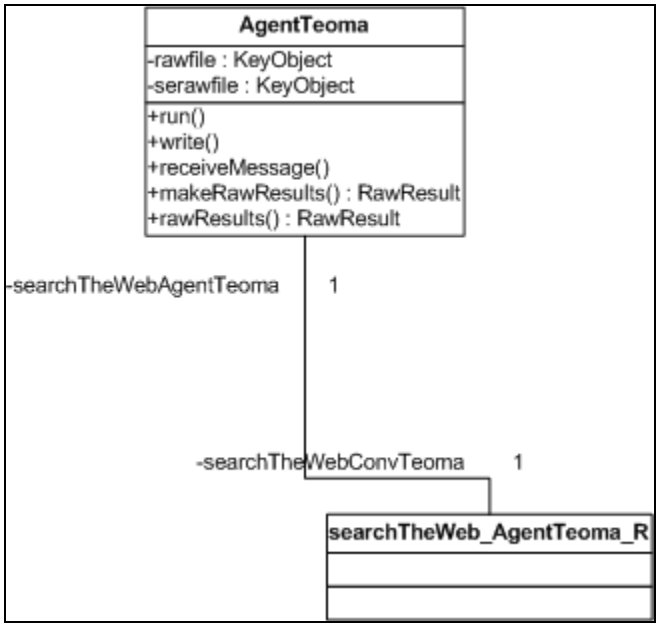


Figure 18 – TeomaPackage

ProducerPackage

ProducerPackage also has two classes called AgentProducer and generateResults_AgentProducer_R. The AgentProducer has a method called generateResult which creates the final HTML file and returns it to the AgentManager. The responder class is used for conversation with the AgentManager.

Figure 19 – ProducerPackage

ManagerPackage

ManagerPackage which acts as the controller for the whole system has several classes namely AgentManager, readUserPreferences_AgentManager_I, rankKeywords_AgentManager_I, searchTheWeb_AgentManager_I and generateResults_AgentManager_I.

The AgentManager creates instances of the conversation initiator classes when it has to get some job done from the other agents in the system. At first it initiates a conversation with the KeyPackage and gets back the keywords typed in the user and also the search preferences. It then initiates a conversation with the GooglePackage and TeomaPackage and requests them to conduct searches based on the keywords and preferences and return the raw results. Once the raw results arrive, the ManaagerPackage calls the ProducerPackage and requests it create a final result in HTML based on all the raw results. The final result is then placed by the Manager in the same computer where the user is typing the article.

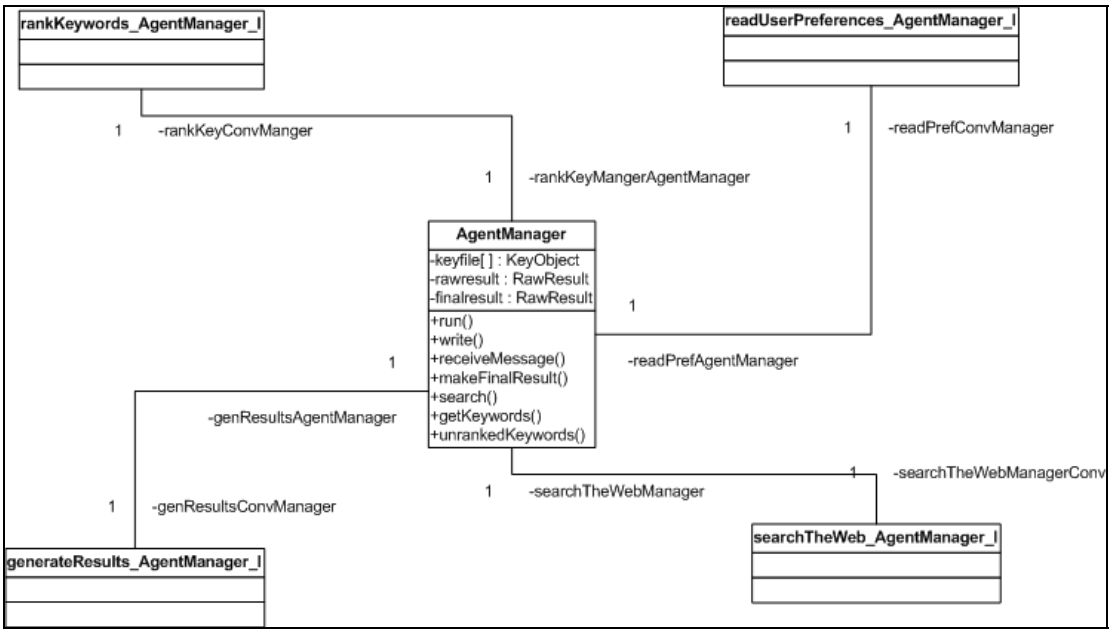


Figure 20 – ManagerPackage

Formal Requirement Specification

A model to test and formally specify a primary property of the MART system was developed. The USE model has four different classes, the most important being the AgentManager. The AgentManager represents the AgentManager class from the system and has access to both the keywords that are generated from the user typed article and the results file created after a search has been conducted.

The KeyObject class in the USE model represents the KeyObject class of the KeyObjectPackage in the Object diagram. The class has three attributes – name, rank and nooftime. These attributes store the keyword itself, the number of times the word appears in the user typed article and a corresponding rank based in comparison to the other keywords.

The RawResult class in the USE model represents the RawResult class in the KeyObjectPackage of the object diagram. In actual implementation the entire result is stored as a string in the RawResult class. However, to show the property of the entire system that some of the original keywords are also present in the result, the USE model has ResultWord classes that are attached to the RawResult class. The ResultWord class represents each and every result words (String) that will be present in the RawResult class in actual implementation.

Constraints

The first constraint "keysalsoinresult" specifies that the search results (RawResult) will have some of the keywords that exist in the keywords (KeyObjects).

The second constraint "rankings" specifies that if a keyword appears more number of times in the user typed article then its rank will be higher than keywords that appear lesser number of times in the typed article.

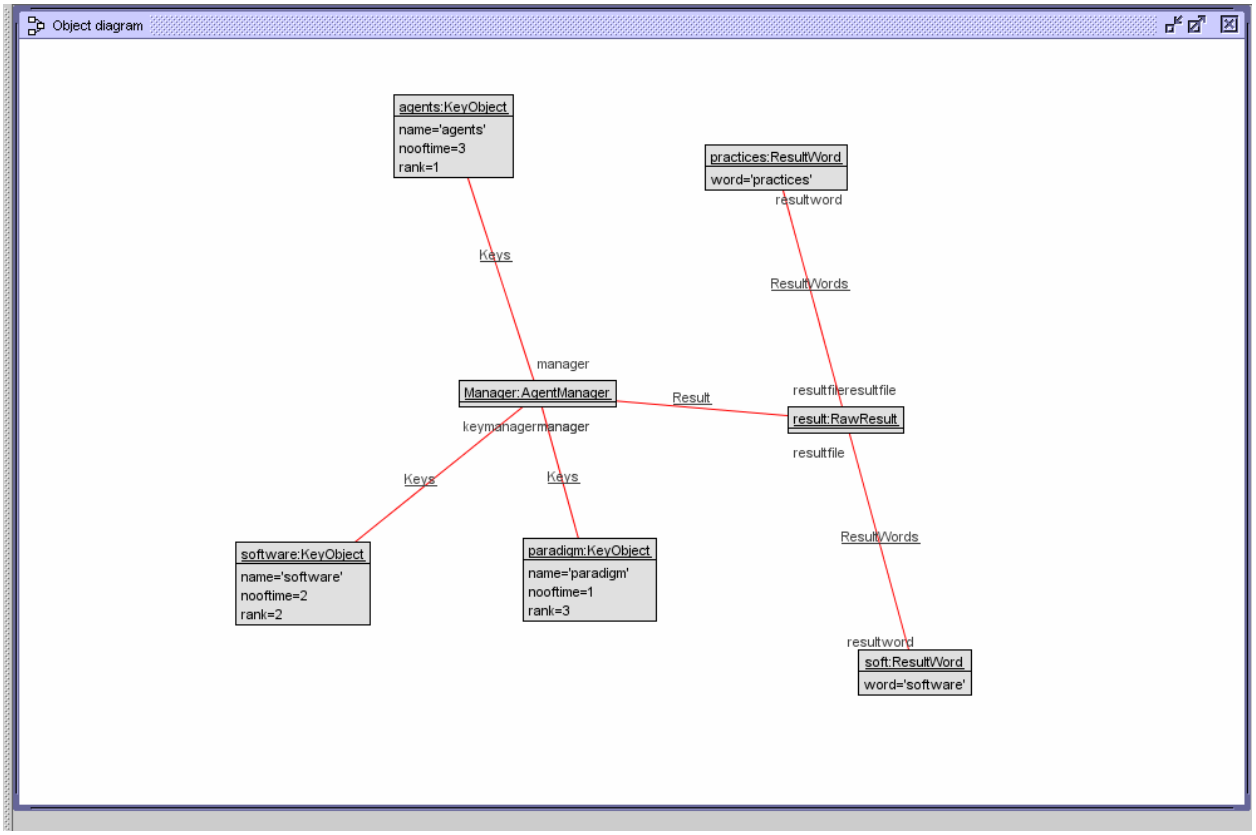


Figure 21 – USE Model

```
-- MART.use in mad-cis\mse\presentation2 folder
-- The model demonstrates the primary property of
-- MART project being built for the MSE project.
-- The property it tries to state and check is that some
-- of the keywords originally typed by the user should also
be
-- present in the final results file
-- Author - Madhukar Kumar
-- Date: July 25, 2003

model MART
--
-- CLASSES --
--
-- The AgentManager agent is the main controller
-- of the entire system
class AgentManager
end

-- Every instance of class KeyObject signifies
-- a single keyword read from a the User typed article
(from MS Word)
class KeyObject
attributes
     name : String
     rank : Integer
     nooftime : Integer
end


class RawResult
end

-- Every instance of ResultWord represents single words
that are found
-- in the RawResult file created by the Sleuth agent. The
RawResult file
-- is created by the Sleuth agent after it has conducted
the search.
-- In actual implementation RawResults will be one single
flat file
-- with several words (String) in it.
class ResultWord
     attributes
          word : String
end
```

```
--
-- ASSOCIATIONS --
--
-- Keys: Manager has several keys attached to it
association Keys between
AgentManager[1] role manager
KeyObject[*] role keys
end

--

-- ResultWords: RawResult has several ResultWord attached
to it
association ResultWords between
RawResult[1] role resultfile
ResultWord[*] role resultword
end

-- Result: KeyManager is attached to several Result File
-- because each sleuth agent generates one RawResult file
association Result between
AgentManager[1] role keymanager
RawResult[1] role resultfile
end



--
-- CONSTRAINTS --
--
constraints

-- Some of the keywords in the KeyManager are also present
-- in the RawResults.

-- Some of the keywords are also present in the RawResult
context km: AgentManager
     inv keysalsoinresult:
          (km.keys.name->asSet)-
>intersection(km.resultfile.resultword.word->asSet)-
>notEmpty


-- If a word appears more number of times in the user
inputted article
-- its ranking will be higher (1 is higher than 2) than a
word which
-- appears lesser number of times in the article
context key : AgentManager
```

```
      inv rankings :
      key.keys->forAll( r,s : KeyObject | r.nooftime > 
s.nooftime implies r.rank < s.rank)
```

CHAPTER 4 – COMPONENT DESIGN

Overview of the system

The overall system has seven major packages as shown in the diagram below. Each package has several classes inside them and their roles and functions are discussed later.

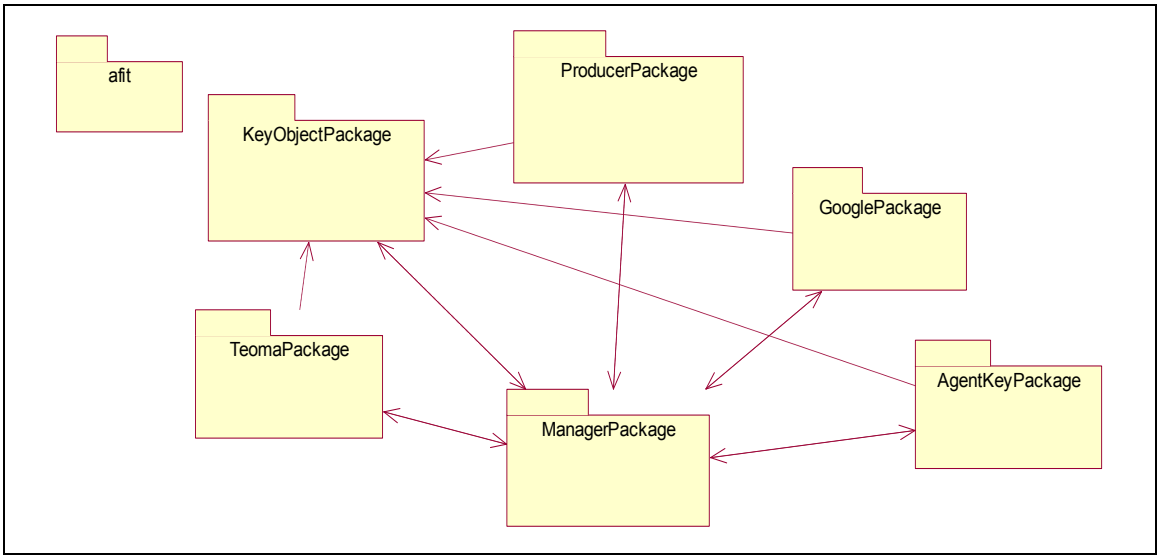


Figure 22- Overview of the system

KeyObjectPackage

The KeyObjectPackage consists of the classes as shown in the next figure. KeyObject is used to store individual keywords with their rank and the number of times it was found in the user typed article. The RawResult object stores the result sent by sleuth agents. It also stores the id of the agent that sent the RawResult object.

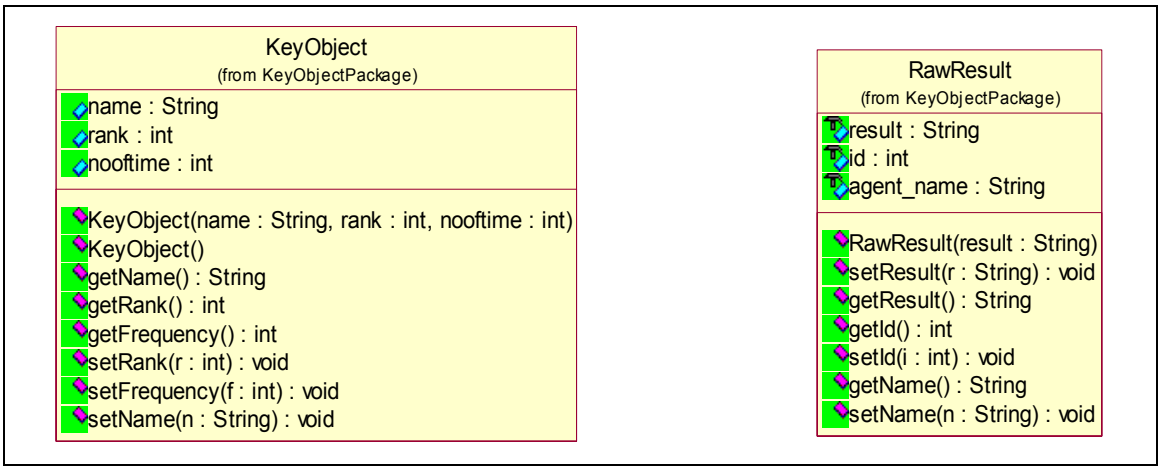


Figure 23 - KeyObjectPackage

KeyObjectPackage Class Documentation

KeyObjectPackage::KeyObject Class Reference

Public Member Functions

- **KeyObject** (String name, int rank, int nooftime)
- String **getName** ()
- int **getRank** ()
- int **getFrequency** ()
- void **setRank** (int r)
- void **setFrequency** (int f)
- void **setName** (String n)

Public Attributes

- String **name**
- int **rank**
- int **nooftime**

Detailed Description

This is a data object used by all the agents

The documentation for this class was generated from the following file:

- KeyObject.java

KeyObjectPackage::RawResult Class Reference

Public Member Functions

- **RawResult** (String result)
- void **setResult** (String r)
- String **getResult** ()
- int **getId** ()
- void **setId** (int i)
- String **getName** ()
- void **setName** (String n)

Package Attributes

- String **result**
- int **id**
- String **agent_name**

Detailed Description

This class is used to transfer result by all Agents

The documentation for this class was generated from the following file:

- RawResult.java

GooglePackage

GooglePackage consists of the following four classes shown in the diagram. The class MulticastSend is a thread that broadcasts the AgentGoogle's name, port number and id to a class D address. When AgentManager is started, it joins the multicast address and receives the agent name, id and port number. Class crawler has one method called doSearch and it is actually responsible for conducting search on the Google search engine using the Google API and a license key issued to the developer. AgentGoogle is the main controller of the system. When it receives a request for conversation from AgentManager, it spawns a new conversation responder thread that controls the conversation. During the conversation, AgentGoogle receives the keywords and based on its current id, it decides how many keywords it should take to conduct a search. Finally when the package has accumulated the result, it returns it as a RawResult object (found in KeyOnjectPackage) to AgentManager's conversation initiator.

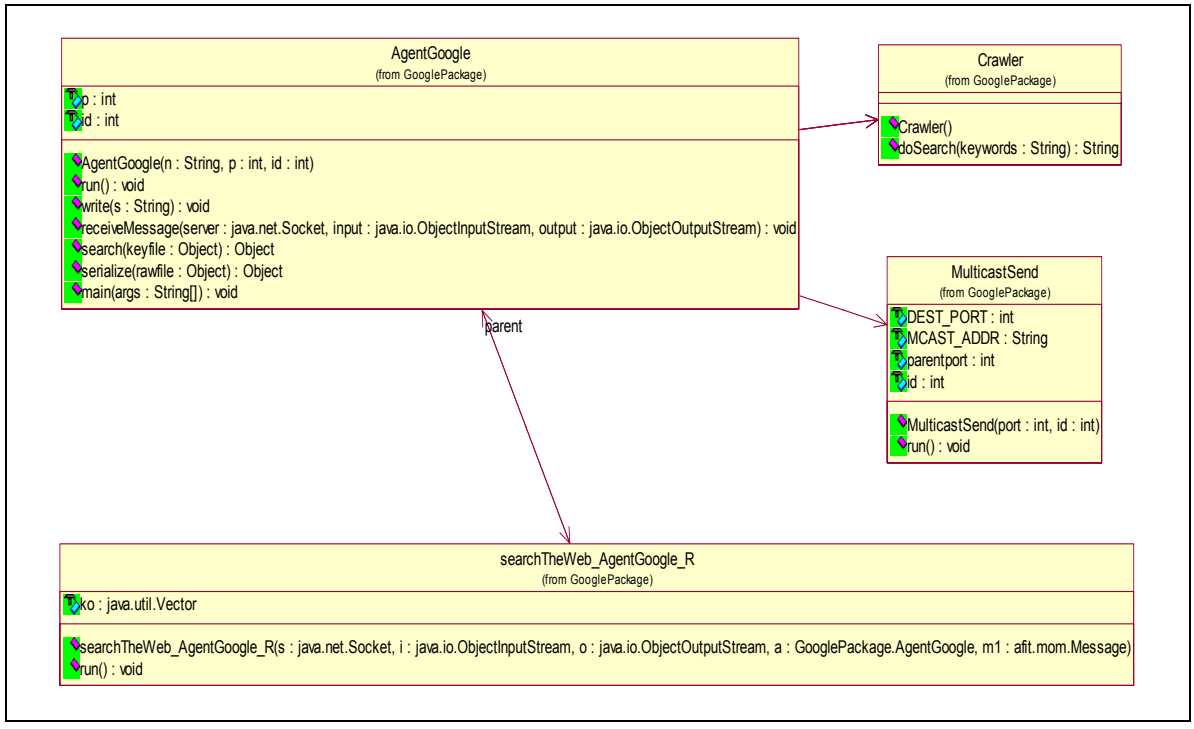


Figure 24 - Class diagram for GooglePackage

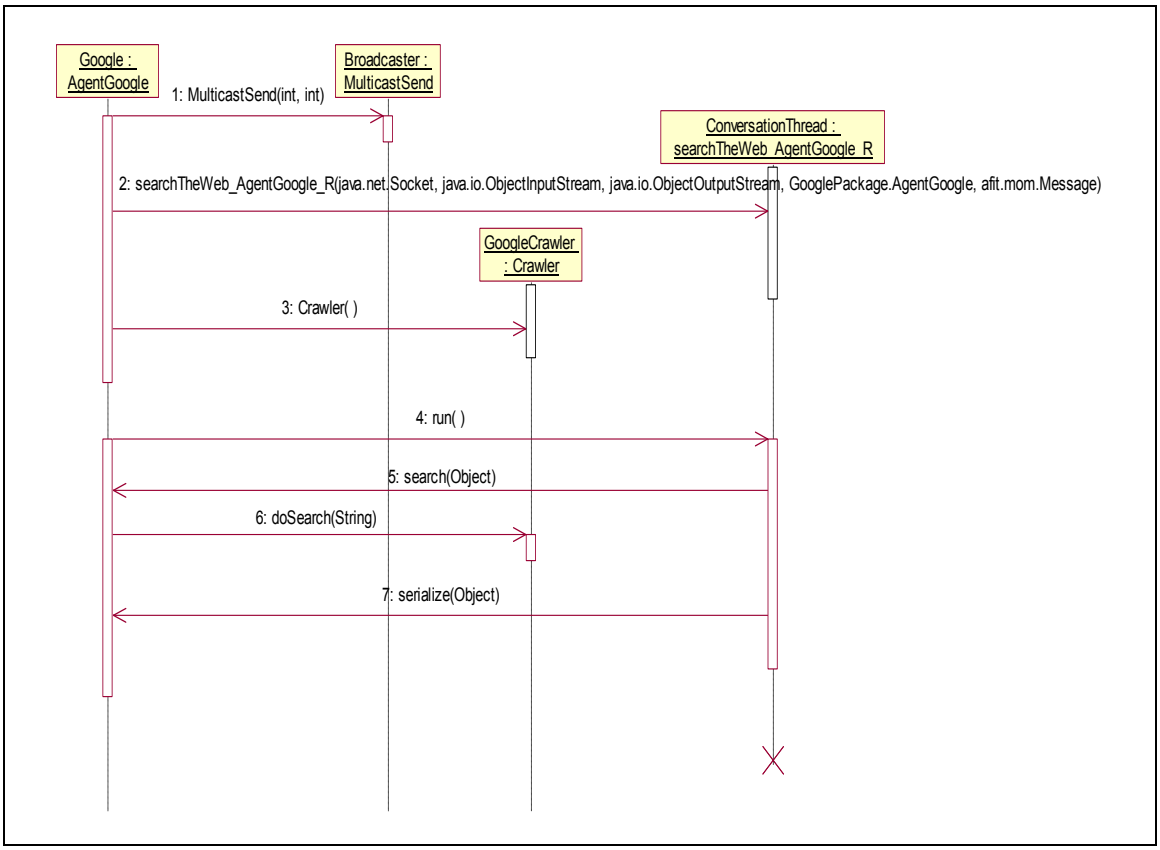


Figure 25 - Sequence diagram for GooglePackage

GooglePackage Class Documentation

GooglePackage::AgentGoogle Class Reference

Public Member Functions

- **AgentGoogle** (String n, int p, int id)
- void **run** ()
- void **write** (String s)
- void **receiveMessage** (Socket server, ObjectInputStream input, ObjectOutputStream output)
- Object **search** (Object keyfile)
- Object **serialize** (Object rawfile)

Static Public Member Functions

- void **main** (String[] args)

Package Attributes

- int **p**
- int **id**

Detailed Description

This agent is responsible for getting search results from the Google website by using the Google API

Constructor & Destructor Documentation

GooglePackage::AgentGoogle::AgentGoogle (String *n*, int *p*, int *id*) [inline]

Constructor merely initializes the agents's name, port number and id

---

Member Function Documentation

void GooglePackage::AgentGoogle::receiveMessage (Socket *server*, ObjectInputStream *input*, ObjectOutputStream *output*) [inline]

This method is called whenever someone tries to start a conversation with **AgentGoogle**. This message checks to see if the performative of the message sent to it is search. If it is then it spawns a new conversation responder thread and passes it the socket and the input and output streams to start conversation If the message performative is not "search" then it returns a sorry message automatically generated by agentTool - agentMom transform ver 0.4

void GooglePackage::AgentGoogle::run () [inline]

Run method for agent Google. Simply outputs a message that it is now running on the specified port number automatically generated by agentTool - agentMom transform ver 0.4

Object GooglePackage::AgentGoogle::search (Object *keyfile*) [inline]

This method first casts the object it has received into a Vector. It then extracts the KeyObjects and checks to see if there is a KeyObject with an id of 9999 or 8888. If

there is one KeyObject with and id of 9999 then the word attached to the object is a website filter that needs to be attached to the end of keywords. If there is an Object with an id of 8888 then the word attached to it (p or d) is the filefilter for pdf or doc file. The method then checks the current AgentGoogle's id. If the current id is 1, then all keywords are extracted, it the id is 2 then top 5 keywords are taken. If the id is 3 then the top 3 keywords are taken. If the id is greater than 3 then only the top most ranking keywords are taken. Once the keywords and the filters are extracted, they are appended to a string that is sent to Crawler's doSearch() method Whatever doSearch() returns, is cast into a RawResult() object and sent back to the caller of this method - the conversation responder automatically generated by agentTool - agentMom transform ver 0.4

void GooglePackage::AgentGoogle::write (String *s*) [inline]

Write method - used to write conversation error messages to output. automatically generated by agentTool - agentMom transform ver 0.4

The documentation for this class was generated from the following file:

- AgentGoogle.java

GooglePackage::Crawler Class Reference

Public Member Functions

- String **doSearch** (String keywords)

Detailed Description

This class actually does the Google search using the Google API and the key provided to the developer. If you are not Madhukar then please sign up at Google for a new API key and copy and paste it below in String s. Currently String s holds the key issued to Madhukar

---

Member Function Documentation

String GooglePackage::Crawler::doSearch (String *keywords*) [inline]

This method actually does the search. It takes a string as a parameter which is used as keywords to search Google

---

The documentation for this class was generated from the following file: Crawler.java

GooglePackage::MulticastSend Class Reference

Public Member Functions

- **MulticastSend** (int port, int id) throws Exception

- void **run** ()

Package Attributes

- int **DEST_PORT**
- String **MCAST_ADDR**
- int **parentport**
- int **id**
- InetAddress **thismachine**

Detailed Description

This thread is like a public address system. It broadcasts a String (holding AgentGoogle's name, id and port number) to a multicast class d address - "225.4.5.6". Anybody who joins this address will be able to read the String sent out by this class. Note - This works only if the receiver is on the same network

---

Member Function Documentation

void GooglePackage::MulticastSend::run () [inline]

The model for the class. This method opens a MulticastSocket() and then continously keeps sending the String out as a message in a never ending loop.

The documentation for this class was generated from the following file: MulticastSend.java

GooglePackage::searchTheWeb_AgentGoogle_R Class Reference

Public Member Functions

- **searchTheWeb_AgentGoogle_R** (Socket s, ObjectInputStream i, ObjectOutputStream o, **AgentGoogle** a, Message m1)
- void **run** ()

Package Attributes

- **AgentGoogle parent**
- Message **gotm**
- Vector **ko**

Detailed Description

Conversation class for **searchTheWeb_AgentGoogle_R**. This thread is responsible for the conversation with AgentManager automatically generated by agentTool - agentMom transform ver 0.4

Constructor & Destructor Documentation

GooglePackage::searchTheWeb_AgentGoogle_R::searchTheWeb_AgentGoogle_R (Socket *s*, ObjectInputStream *i*, ObjectOutputStream *o*, AgentGoogle *a*, Message

***m1*) [inline]**

Constructor for **searchTheWeb_AgentGoogle_R** merely initializes the variables it receives from the **AgentGoogle** automatically generated by agentTool - agentMom transform ver 0.4

Member Function Documentation

void GooglePackage::searchTheWeb_AgentGoogle_R::run () [inline]

Conversation run method controls the conversation with AgentManager. This method calls the AgentGoogle's search() method and collects the result which it wraps around a message object and sends back to AgentManager's conversation initiator. automatically generated by agentTool - agentMom transform ver 0.4

The documentation for this class was generated from the following file: searchTheWeb_AgentGoogle_R.java

TeomaPackage

The TeomaPackage consists of the classes shown in the next figure. When AgentTeoma receives a conversation request with a search performative, it starts the conversation responder thread. The conversation responder thread – searchTheWeb_AgentTeoma_R class then invokes the search method in AgentTeoma that crawls on a search engine with the keywords received from AgentManager and

returns the result wrapped in a RawResult object. The MulticastSend object behaves identically to the one described under GooglePackage.

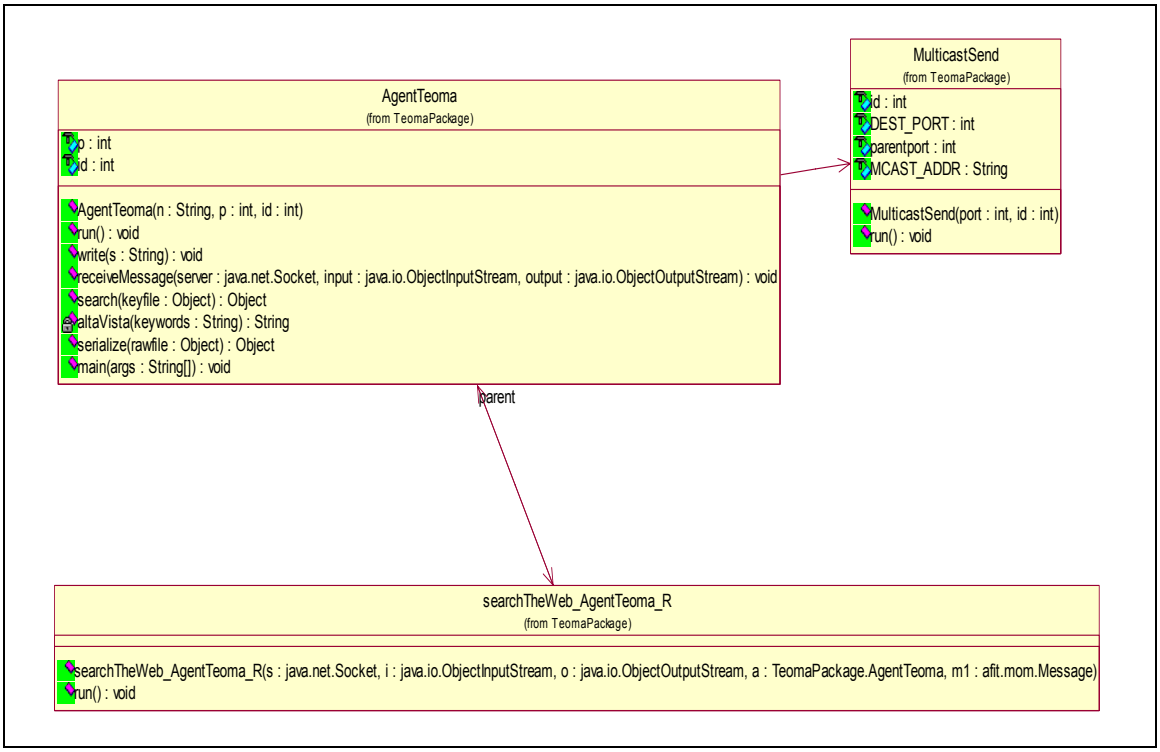

AgentTeoma
(from TeomaPackage)
p : int
id : int
AgentTeoma(n : String, p : int, id : int)
run() : void
write(s : String) : void
receiveMessage(server : java.net.Socket, input : java.io.ObjectInputStream, output : java.io.ObjectOutputStream) : void
search(keyfile : Object) : Object
altaVista(keywords : String) : String
serialize(rawfile : Object) : Object
main(args : String[]) : void
MulticastSend
(from TeomaPackage)
id : int
DEST_PORT : int
parentport : int
MCAST_ADDR : String
MulticastSend(port : int, id : int)
run() : void
parent
searchTheWeb_AgentTeoma_R
(from TeomaPackage)
searchTheWeb_AgentTeoma_R(s : java.net.Socket, i : java.io.ObjectInputStream, o : java.io.ObjectOutputStream, a : TeomaPackage.AgentTeoma, m1 : afit.mom.Message)
run() : void

Figure 26 - TeomaPackage

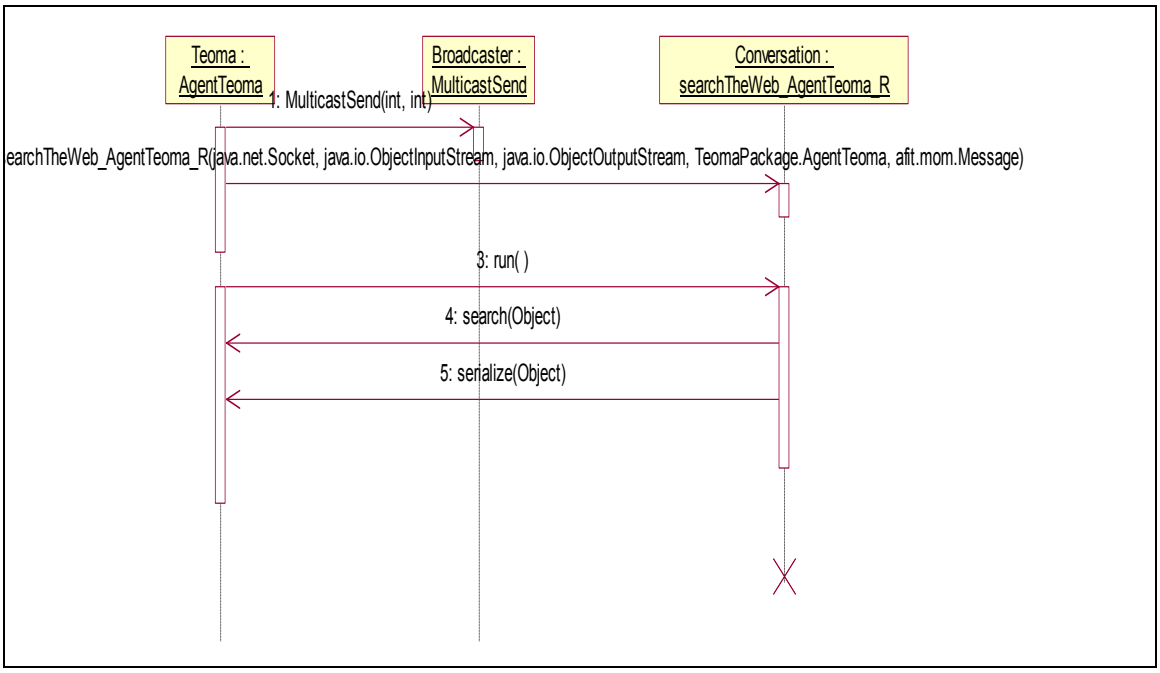


Figure 27 - Sequence diagram for TeomaPackage

TeomaPackage Class Documentation

TeomaPackage::AgentTeoma Class Reference

Public Member Functions

- **AgentTeoma** (String n, int p, int id)
- void **run** ()
- void **write** (String s)
- void **receiveMessage** (Socket server, ObjectInputStream input, ObjectOutputStream output)
- Object **search** (Object keyfile)
- Object **serialize** (Object rawfile)

Static Public Member Functions

- void **main** (String[] args)

Package Attributes

- int **p**
- int **id**

Detailed Description

AgentTeoma crawls on a search engine and returns with the search results automatically generated by agentTool - agentMom transform ver 0.4

Constructor & Destructor Documentation

TeomaPackage::AgentTeoma::AgentTeoma (String *n*, int *p*, int *id*) [inline]

Constructor for agent. Note - Agent tool has a bug which is why the developer has to manually create and start an instance of MessageHandler inside the Agent constructor. The constructor also creates and starts an instance of **MulticastSend** automatically generated by agentTool - agentMom transform ver 0.4

Member Function Documentation

void TeomaPackage::AgentTeoma::receiveMessage (Socket *server*, ObjectInputStream *input*, ObjectOutputStream *output*) [inline]

Required method for agentMom framework. Whenever someone tries to start a conversation with this Agent, it checks to see if the message performative is "search". If it is then a new conversation responder thread is started otherwise a sorry message is sent back automatically generated by agentTool - agentMom transform ver 0.4

void TeomaPackage::AgentTeoma::run () [inline]

Run method for agent merely prints a statement that the Agent is now successfully running at the specified port number automatically generated by agentTool - agentMom transform ver 0.4

Object TeomaPackage::AgentTeoma::search (Object *keyfile*) [inline]

Method rawfile=search referenced in conversation. The method looks at the current Agent's id and then decides how many keywords will be extracted for search. It then calls a private method called alttVista that actually conducts the search on the search engine automatically generated by agentTool - agentMom transform ver 0.4

Object TeomaPackage::AgentTeoma::serialize (Object *rawfile*) [inline]

Method serawfile=serialize referenced in conversation . This method takes an Object as a parameter. It casts the Object to a String and then wraps it around a RawResult object before sending it back to the caller automatically generated by agentTool - agentMom transform ver 0.4

void TeomaPackage::AgentTeoma::write (String *s*) [inline]

Write method - used to write conversation error messages to output. automatically generated by agentTool - agentMom transform ver 0.4

The documentation for this class was generated from the following file:

- AgentTeoma.java

TeomaPackage::MulticastSend Class Reference

Public Member Functions

- **MulticastSend** (int port, int id) throws Exception
- void **run** ()

Package Attributes

- int **id**
- int **DEST_PORT**
- int **parentport**
- String **MCAST_ADDR**

Detailed Description

This class is identical to **MulticastSend** of GooglePackage Please refer to GooglePackage documentation for more information about this class

---

The documentation for this class was generated from the following file: MulticastSend.java

TeomaPackage::searchTheWeb_AgentTeoma_R Class Reference

Public Member Functions

- **searchTheWeb_AgentTeoma_R** (Socket s, ObjectInputStream i, ObjectOutputStream o, **AgentTeoma** a, Message m1)
- void **run** ()

Package Attributes

- **AgentTeoma parent**
- Message **gotm**

Detailed Description

Conversation class for **searchTheWeb_AgentTeoma_R**. This class manages the conversation with AgentManager's conversation initiator and finally returns the search result which it gets from AgentTeoma's search() method automatically generated by agentTool - agentMom transform ver 0.4

Constructor & Destructor Documentation

TeomaPackage::searchTheWeb_AgentTeoma_R::searchTheWeb_AgentTeoma_R (Socket *s*, ObjectInputStream *i*, ObjectOutputStream *o*, AgentTeoma *a*, Message *m1*) [inline]

Constructor for **searchTheWeb_AgentTeoma_R**. Note that AgentTool has a bug. The developer has to ensure that the Message m it receives in the parameter is the same that is used in the run method. AgentToo generates code in which there are two different Messages and hence this class will not run if the above bug is not fixed by the developer automatically generated by agentTool - agentMom transform ver 0.4

Member Function Documentation

void TeomaPackage::searchTheWeb_AgentTeoma_R::run () [inline]

Conversation run method controls the conversation with AgentManager's conversation initiator. It receives a search message with the keywords. The keywords String is sent to the search() method in **AgentTeoma** that returns the search result. The search result is finally sent back to the AgentManager automatically generated by agentTool - agentMom transform ver 0.4

The documentation for this class was generated from the following file: searchTheWeb_AgentTeoma_R.java

AgentKeyPackage

This package has three main classes as shown in the next figure. It has the GUI user interface UserPref.java that shows up when the user clicks on FilteredSearch button from MS Word. The NewPref object is the class that handles the event from the OK button push event of the UserPref class. The AgentKey has two main methods that are responsible for extracting keywords and making KeyObjects out of them and also for ranking the keywords.

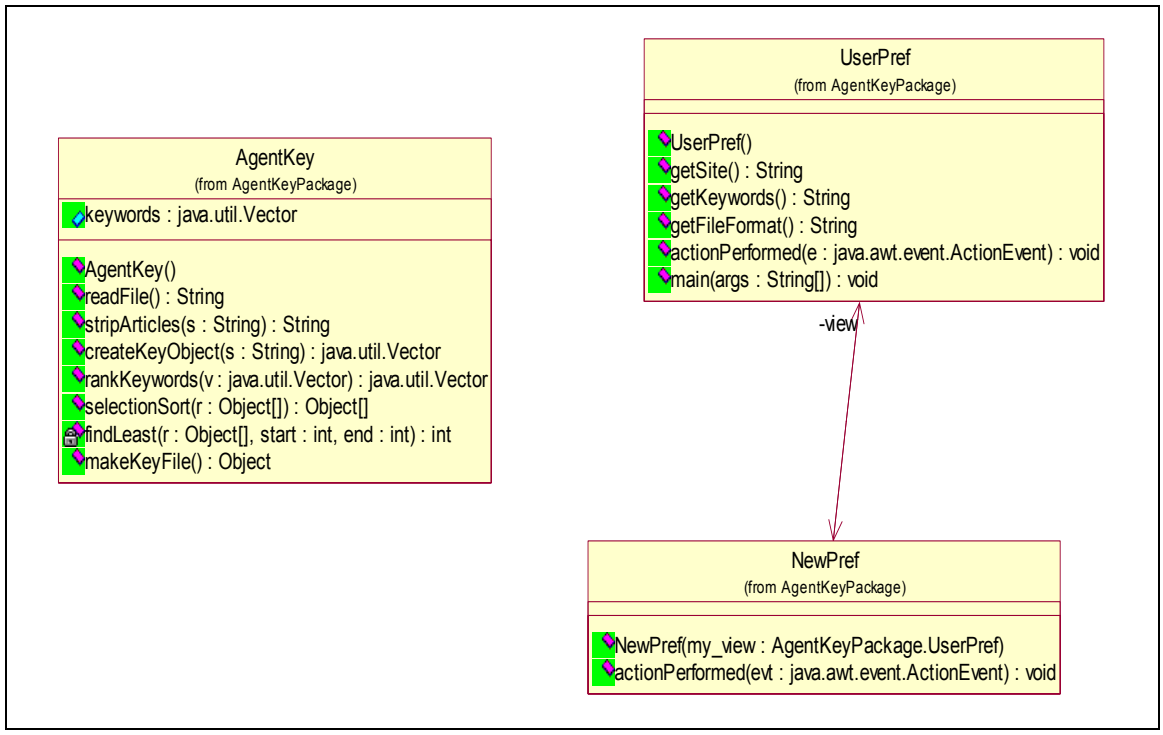


Figure 28 - AgentKeyPackage

AgentKeyPackage Class Documentation

AgentKeyPackage::AgentKey Class Reference

Public Member Functions

- **AgentKey** ()
- String **readFile** ()

- String **stripArticles** (String s)
- Vector **createKeyObject** (String s)
- Vector **rankKeywords** (Vector v)
- Object[] **selectionSort** (Object[] r)

Public Attributes

- Vector **keywords**

Detailed Description

This is not like the rest of the agents in the sense it does not converse with any other agent but the Manager. The methods here are all called by AgentManager. This class reads a txt file and filters out words like "of," "a" etc then creates a KeyObject for each word. Finally it adds the ranking and nooftime in each object and returns a Vector of the KeyObjects to the Manager

---

Constructor & Destructor Documentation

AgentKeyPackage::AgentKey::AgentKey () [inline]

Just initializes the Vector which will be filled with KeyObjectss

---

Member Function Documentation

Vector AgentKeyPackage::AgentKey::createKeyObject (String *s*) [inline]

This method takes a string as a parameter. It takes the string, breaks them into individual words makes KeyObject of each word and adds them to the vector - keywords. It returns the Vector or all KeyObjects when it finishes. Algorithm -- First check if the word exists in the keyword, if it does then merely increment its nooftime attribute by 1. If it does not then just create a new KeyObject and add to the vector. At this point all ranks are 0 (invalid). The ranking will be done by **rankKeywords()**

Vector AgentKeyPackage::AgentKey::rankKeywords (Vector *v*) [inline]

rankMethod takes the Vector of KeyObjects and arranges everything in decending order based on the frequency (nooftime) of each keywords. It then takes the sorted vector and sets the rank for each one of them by giving 1 to top most 2 to next and so on Algorithm - It takes the vector, converts it to an array and calls the selectionSort method Once it receives the vector, it reconverts the array to a vector, adds the ranks and returns the vector.

String AgentKeyPackage::AgentKey::readFile () [inline]

This method merely reads the keywords.txt file saved by the word Macro and returns everything as a string.

Object [] AgentKeyPackage::AgentKey::selectionSort (Object[] *r*) [inline]

This is a reusable method. It takes an array as a parameter and returns a sorted array to its called. This method uses findLeast for its internal computation

String AgentKeyPackage::AgentKey::stripArticles (String *s*) [inline]

This method takes a sting as a parameter It goes through the string and stips out all the irrelevant words that will not be used for ranking or searching.

The documentation for this class was generated from the following file: AgentKey.java

AgentKeyPackage::NewPref Class Reference

Public Member Functions

- **NewPref** (**UserPref** my_view)
- void **actionPerformed** (ActionEvent evt)

Detailed Description

This class handles the OK button push action from the **UserPref** GUI class.

Constructor & Destructor Documentation

AgentKeyPackage::NewPref::NewPref (UserPref *my_view*) [inline]

Constructor - Does nothing

Member Function Documentation

void AgentKeyPackage::NewPref::actionPerformed (ActionEvent *evt*) [inline]

actionPerformed handles the OK button-push event This method extracts the user input from the fields and checks to see whether some field is non empty Finally, it collects the input and calls the forNewPref method in the AgentManager. The forNewPref method calls the search, make results etc

---

The documentation for this class was generated from the following file: NewPref.java

AgentKeyPackage::UserPref Class Reference

Public Member Functions

- **UserPref** ()
- String **getSite** ()
- String **getKeywords** ()
- String **getFileFormat** ()
- void **actionPerformed** (ActionEvent e)

Static Public Member Functions

- void **main** (String[] args)

Detailed Description

UserPref displays a frame to the user and gets preferences If the user clicks on the OK button, the class **NewPref** handles the event.

Constructor & Destructor Documentation

AgentKeyPackage::UserPref::UserPref () [inline]

Constructor creates a frame and waits for an event

Member Function Documentation

void AgentKeyPackage::UserPref::actionPerformed (ActionEvent *e*) [inline]

This method handles Cancel button - Graceful exit

String AgentKeyPackage::UserPref::getFileFormat () [inline]

This method merely extracts what the user typed in the fileformat textfield

String AgentKeyPackage::UserPref::getKeywords () [inline]

This method merely extracts what the user typed in the keywords textfield

String AgentKeyPackage::UserPref::getSite () [inline]

This method merely extracts what the user typed in the site textfield

void AgentKeyPackage::UserPref::main (String[] *args*) [inline, static]

Create a new instance of **UserPref()**

The documentation for this class was generated from the following file: UserPref.java

ProducerPackage

The ProducerPackage has three classes as shown in the following figure. The MulticastSend object behaves identically to the one in the GooglePackage. When AgentProducer receives a request for conversation, it starts a conversation responder thread. The conversation responder thread receives a Vector of RawResults which it passes on to AgentProducer. The AgentProducer takes the Vector, extracts the results and based on the ids of the agents that sent the result, it produces a list of relevant results and wraps HTML around it. It then sends the HTML wrapped String to the conversation responder, which in turn sends back the final result to the AgentManager.

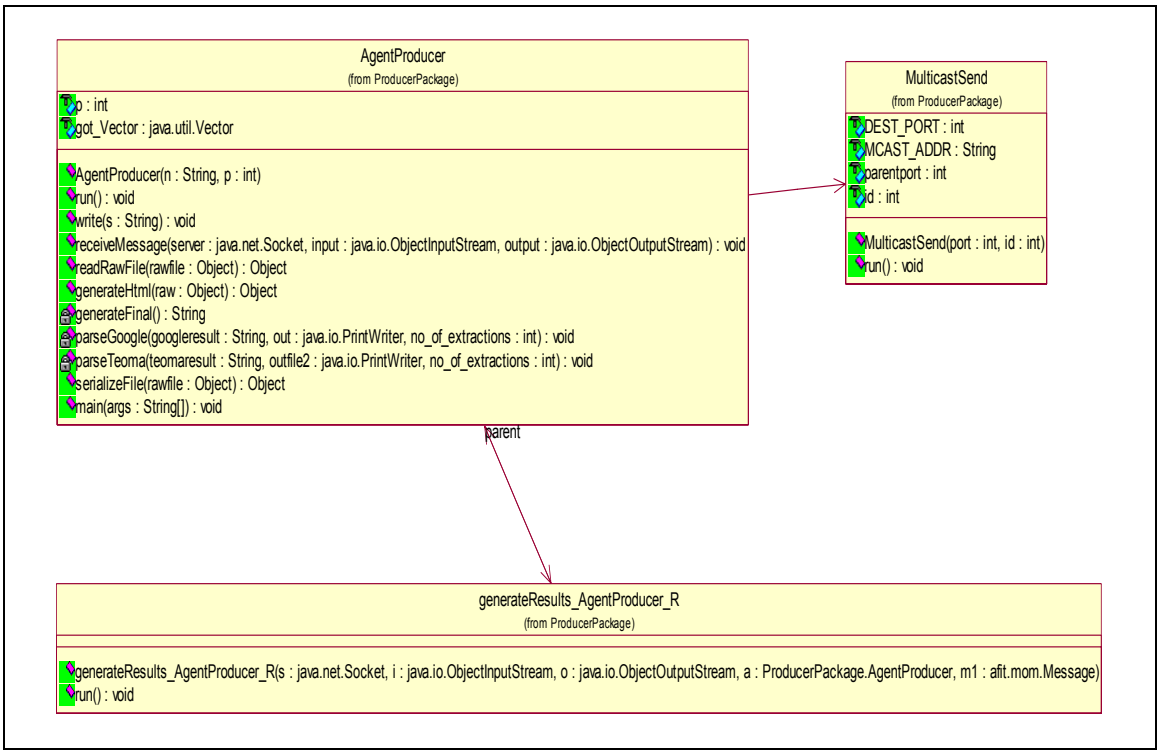

AgentProducer
(from ProducerPackage)
p : int
got_Vector : java.util.Vector
AgentProducer(n : String, p : int)
run() : void
write(s : String) : void
receiveMessage(server : java.net.Socket, input : java.io.ObjectInputStream, output : java.io.ObjectOutputStream) : void
readRawFile(rawfile : Object) : Object
generateHtml(raw : Object) : Object
generateFinal() : String
parseGoogle(googleresult : String, out : java.io.PrintWriter, no_of_extractions : int) : void
parseTeoma(teomaresult : String, outfile2 : java.io.PrintWriter, no_of_extractions : int) : void
serializeFile(rawfile : Object) : Object
main(args : String[]) : void
MulticastSend
(from ProducerPackage)
DEST_PORT : int
MCAST_ADDR : String
parentport : int
id : int
MulticastSend(port : int, id : int)
run() : void
parent
generateResults_AgentProducer_R
(from ProducerPackage)
generateResults_AgentProducer_R(s : java.net.Socket, i : java.io.ObjectInputStream, o : java.io.ObjectOutputStream, a : ProducerPackage.AgentProducer, m1 : afit.mom.Message)
run() : void

Figure 29 - ProducerPackage

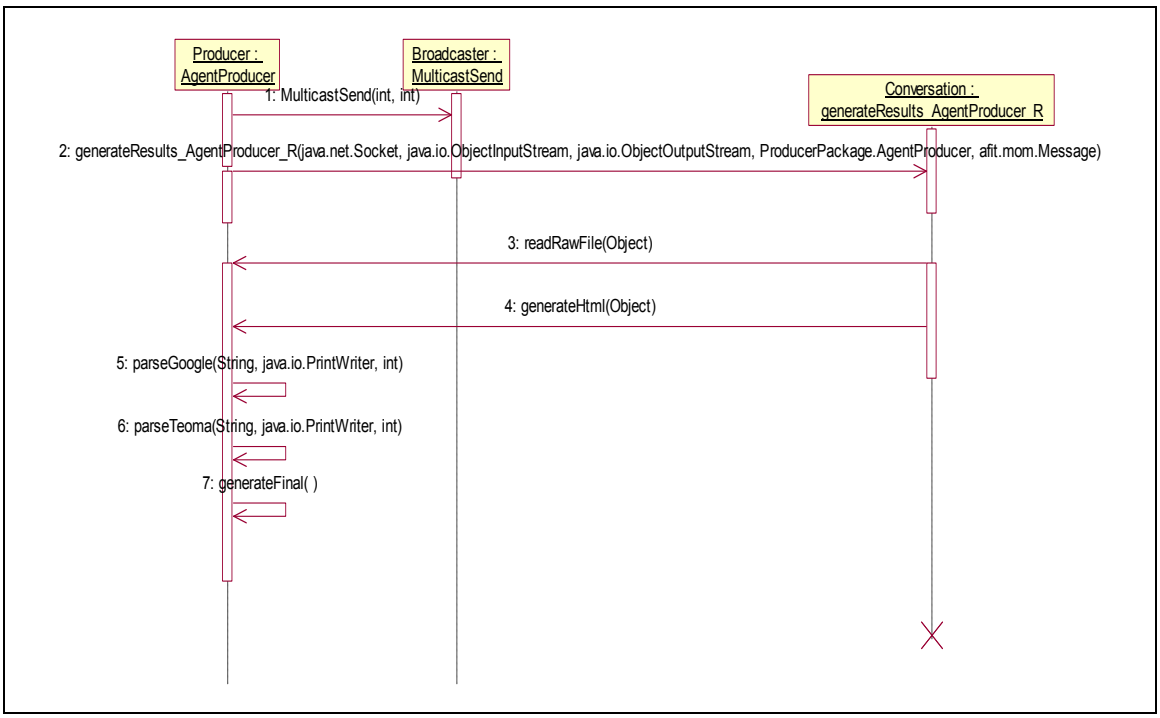


Figure 30 - AgentProducer sequence diagram

ProducerPackage Class Documentation

ProducerPackage::AgentProducer Class Reference

Public Member Functions

- **AgentProducer** (String n, int p)
- void **run** ()
- void **write** (String s)
- void **receiveMessage** (Socket server, ObjectInputStream input, ObjectOutputStream output)
- Object **readRawFile** (Object rawfile)
- Object **generateHtml** (Object raw)

Static Public Member Functions

- void **main** (String[] args)

Package Attributes

- int **p**
- Vector **got_Vector**

Detailed Description

Class **AgentProducer** gets a vector of RawResults. It filters out the result and returns a string that is written to a file by AgentManager and results in the final_result.html file that is shown to the user automatically generated by agentTool - agentMom transform ver 0.4

Constructor & Destructor Documentation

ProducerPackage::AgentProducer::AgentProducer (String *n*, int *p*) [inline]

Constructor for agent. automatically generated by agentTool - agentMom transform ver 0.4

Member Function Documentation

Object ProducerPackage::AgentProducer::generateHtml (Object *raw*) [inline]

Method generateHtml, referenced in conversation, is actually responsible for creating the final html string. It takes all the RawResult objects one by one and sees if it was sent by Teoma or Google Agent. Based on who sent the result, the string is parsed by calling either parseGoogle() or parseTeoma() methods. Finally, it checks to see the id of the agents that sent the result and picks only certain amount of results and adds them to the final result string. The html result string is finally cast into a serializable object before it is sent back automatically generated by agentTool - agentMom transform ver 0.4

Object ProducerPackage::AgentProducer::readRawFile (Object *rawfile*) [inline]

Method readRawFile, referenced in conversation, merely casts the object that it receives, into a Vector and then extracts all the elements of the vector and casts them into RawResult object

void ProducerPackage::AgentProducer::receiveMessage (Socket *server*, ObjectInputStream *input*, ObjectOutputStream *output*) [inline]

This method is called when someone tries to start a conversation with **AgentProducer** This method checks to see if the attempt is valid and if it is then it spawns a new conversation responder thread by passing the values of socket and the input and the output streams

---

The documentation for this class was generated from the following file: AgentProducer.java

ProducerPackage::generateResults_AgentProducer_R Class Reference

Public Member Functions

- **generateResults_AgentProducer_R** (Socket s, ObjectInputStream i, ObjectOutputStream o, **AgentProducer** a, Message m1)
- void **run** ()

Package Attributes

- **AgentProducer parent**
- Message **gotm**

Detailed Description

Conversation class for **generateResults_AgentProducer_R** is responsible for carrying out the conversation with AgentManager and sending back the final result. automatically generated by agentTool - agentMom transform ver 0.4

Constructor & Destructor Documentation

ProducerPackage::generateResults_AgentProducer_R::generateResults_AgentProducer_R (Socket *s*, ObjectInputStream *i*, ObjectOutputStream *o*, AgentProducer *a*, Message *m1*) [inline]

Constructor for **generateResults_AgentProducer_R** merely initializes all the variables automatically generated by agentTool - agentMom transform ver 0.4

Member Function Documentation

void ProducerPackage::generateResults_AgentProducer_R::run () [inline]

Conversation run method - controls the actual conversation. This method calls Agent Producer's readRawFile() and generateHtml() method and returns the final result to AgentManager. automatically generated by agentTool - agentMom transform ver 0.4

The documentation for this class was generated from the following file: generateResults_AgentProducer_R.java

ProducerPackage::MulticastSend Class Reference

Public Member Functions

- **MulticastSend** (int port, int id) throws Exception
- void **run** ()

Package Attributes

- int **DEST_PORT**
- String **MCAST_ADDR**
- int **parentport**
- int **id**
- InetAddress **thismachine**

Detailed Description

This class is identical to the **MulticastSend** class of GooglePackage Please refer to the GooglePackage documentation

---

The documentation for this class was generated from the following file: MulticastSend.java

ManagerPackage

The ManagerPackage acts as the main controller of the system and has the following five main classes as shown in the figure below. When AgentManager gets started, it creates a new instance of MulticastReceive. MulticastReceive joins a predefined multicast address and finds out how many AgentGoogle and AgentTeoma are running. For each unique agent discovered, MulticastReceive creates a RunningAgent object and stores the agents name, id and port number. It then stores all the RunningAgents object in a Vector and sends the Vector back to AgentManager. The MulticastReceiver also records the port number and machine IP address of a running AgentProducer and passes it on to AgentManager.

AgentManager now contacts the AgentKeyPackage and gets back a Vector of KeyObjects. It then sends the Vector to the running instances of AgentGoogle and AgentTeoma. It gets back RawResult object from each sleuth agent that it stores in a vector. AgentManager now sends the vector of RawResult to AgentProducer. AgentProducer in turn makes a list of the final result and encapsulates it with html headers and footers. When AgentManager gets this String, it merely writes it to a file called final_result.html in the c:\mart folder.

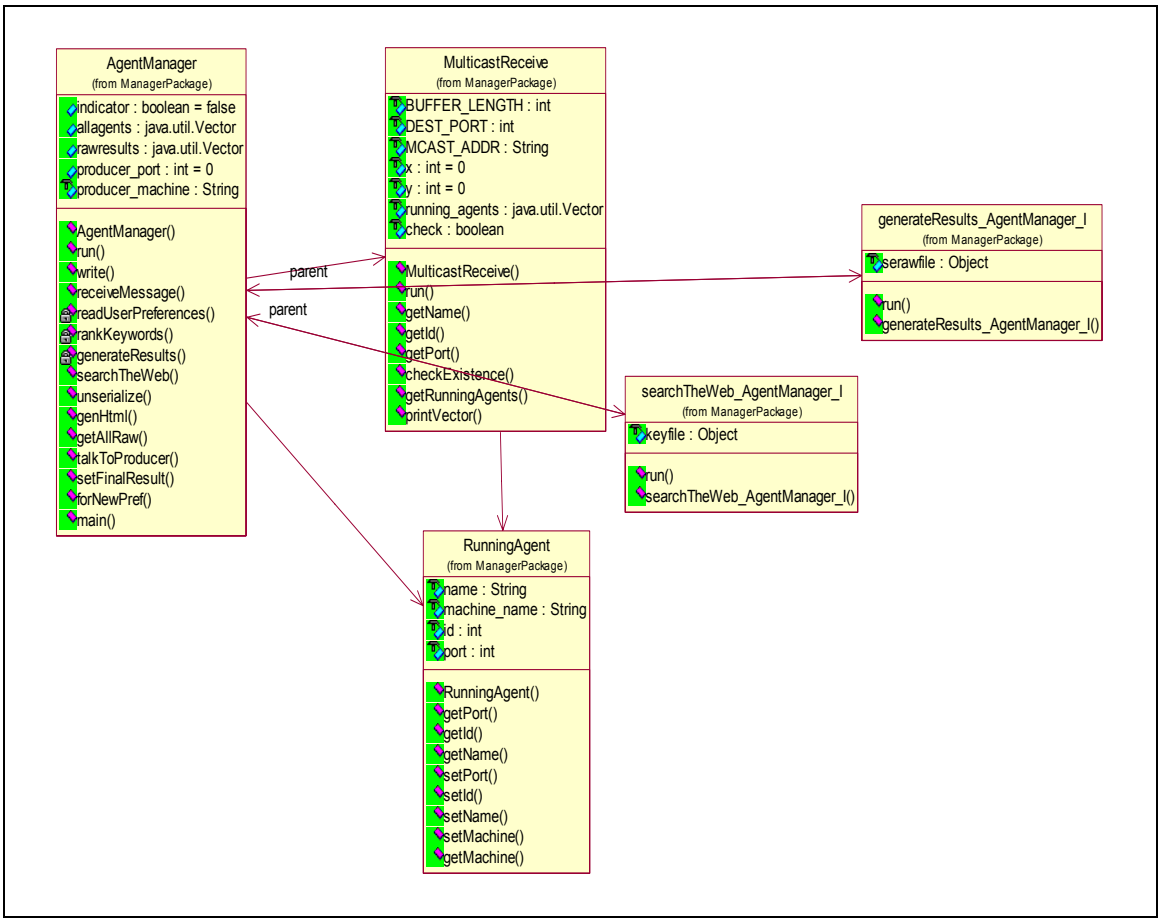


Figure 31 - ManagerPackage

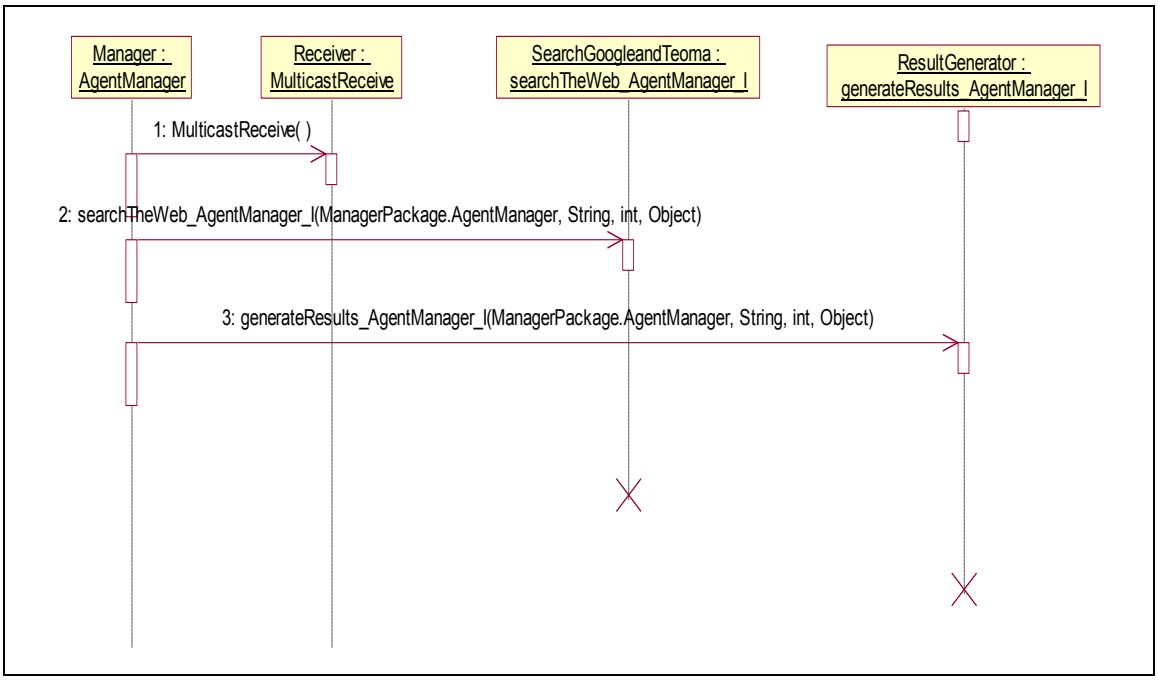


Figure 32 - ManagerPackage sequence diagram

ManagerPackage Namespace Documentation

KeyObjectPackage Namespace Reference

Detailed Description

This class is the Controller of the entire system. It resides on the same machine on which MS Word is running and also on which the AgentKey (since AgentKey needs to read what the user has already typed in MS Word) is running.

Author:

- Madhukar Kumar September 16, 2003 - ready for presentation 3

ManagerPackage Class Documentation

ManagerPackage::generateResults_AgentManager_I Class Reference

Public Member Functions

- **generateResults_AgentManager_I** (AgentManager a, String hostName, int portNum, Object serawfile_in)
- void **run** ()

Package Attributes

- AgentManager **parent**
- Object **serawfile**

Detailed Description

Conversation class for **generateResults_AgentManager_I**. This class is responsible for handling the conversation with AgentProducer automatically generated by agentTool - agentMom transform ver 0.4

Constructor & Destructor Documentation

ManagerPackage::generateResults_AgentManager_I::generateResults_AgentManager_I (AgentManager *a*, String *hostName*, int *portNum*, Object *serawfile_in*) [inline]

Constructor for **generateResults_AgentManager_I**. automatically generated by agentTool - agentMom transform ver 0.4

Member Function Documentation

void ManagerPackage::generateResults_AgentManager_I::run () [inline]

Conversation run method controls the conversation with AgentProducer. The method sends a Vector of RawResults and gets back an html wrapped String. This String is passed on to AgentManager through its setFinalResult() method automatically generated by agentTool - agentMom transform ver 0.4

The documentation for this class was generated from the following file:

- generateResults_AgentManager_I.java

ManagerPackage::MulticastReceive Class Reference

Public Member Functions

- **MulticastReceive** ()
- void **run** ()
- String **getName** (String m)
- int **getId** (String m)
- int **getPort** (String m)
- boolean **checkExistence** (String n, int x, int y)

- Vector **getRunningAgents** ()
- void **printVector** ()

Package Attributes

- int **BUFFER_LENGTH**
- int **DEST_PORT**
- String **MCAST_ADDR**
- int **x** = 0
- int **y** = 0
- Vector **running_agents**
- boolean **check**

Detailed Description

This class is responsible for finding out how many Sleuth agents and AgentProducer is running and what are their port numbers. The class creats a **RunningAgent** object for each Agent that it discovers, and adds them to a Vector. This Vector is finally sent back to AgentManager

---

Constructor & Destructor Documentation

ManagerPackage::MulticastReceive::MulticastReceive () [inline]

The constructor sets the multicast address to 225.4.5.6 since this and port 5000 since this is the combination used by the MulticastSenders to broadcast their messages.

Member Function Documentation

boolean ManagerPackage::MulticastReceive::checkExistence (String *n*, int *x*, int *y*) [inline]

This method is also used by **run()**. It checks to see if an agent is already stored in the vector If not then add it to vector otherwise ignore (do nothing)

int ManagerPackage::MulticastReceive::getId (String *m*) [inline]

A simple method to extract the agentid from the string passed on to it as a parameter. Returns an int This method is called by the **run()** method

String ManagerPackage::MulticastReceive::getName (String *m*) [inline]

A simple method to extract the name from the string passed on to it as a parameter This method is called by the **run()** method

int ManagerPackage::MulticastReceive::getPort (String *m*) [inline]

A simple method to extract the agentid from the string passed on to it as a parameter. Returns an int This method is called by the **run()** method

void ManagerPackage::MulticastReceive::printVector () [inline]

Debug method Please ignore

void ManagerPackage::MulticastReceive::run () [inline]

This is where the thread polls on the classD address and each time it spots a unique MulticastSend message, it extracts the agent name, id and port numbers from it, creates a **RunningAgent** object from it and inserts it into the Vecot called running_agents.

The documentation for this class was generated from the following file: MulticastReceive.java

ManagerPackage::RunningAgent Class Reference

Public Member Functions

- **RunningAgent** (String name, int id, int port)
- int **getPort** ()
- int **getId** ()
- String **getName** ()
- void **setPort** (int p)
- void **setId** (int i)
- void **setName** (String n)
- void **setMachine** (String mac)

- String **getMachine** ()

Package Attributes

- String **name**
- String **machine_name**
- int **id**
- int **port**

Detailed Description

This is a simple class to store an agent discovered by MulticastReceiver. This class stores the agent's name, id, port number and has simple get and set methods for all the above attributes

---

The documentation for this class was generated from the following file:

- RunningAgent.java

ManagerPackage::searchTheWeb_AgentManager_I Class Reference

Public Member Functions

- **searchTheWeb_AgentManager_I** (AgentManager a, String hostName, int portNum, Object keyfile_in)
- void **run** ()

Package Attributes

- AgentManager **parent**
- Object **keyfile**

Detailed Description

Conversation class for **searchTheWeb_AgentManager_I**. automatically generated by agentTool - agentMom transform ver 0.4 This class starts a conversation with AgentGoogles and AgentTeomas

---

Constructor & Destructor Documentation

ManagerPackage::searchTheWeb_AgentManager_I::searchTheWeb_AgentManager_I (AgentManager *a*, String *hostName*, int *portNum*, Object *keyfile_in*) [inline]

Constructor for **searchTheWeb_AgentManager_I**. automatically generated by agentTool - agentMom transform ver 0.4

Member Function Documentation

void ManagerPackage::searchTheWeb_AgentManager_I::run () [inline]

Conversation run method - controls conversation. automatically generated by agentTool - agentMom transform ver 0.4 It sends the Vector of KeyObjects and gets back a RawResult object from the Sleuth agents.

The documentation for this class was generated from the following file:

- searchTheWeb_AgentManager_I.java

CHAPTER 5 – SOFTWARE QUALITY ASSURANCE

Purpose

The purpose of the documents is to define the Software Quality Assurance (SQA) plan that will be adopted throughout the project to maintain an acceptable quality of the final product.

References

1. Vision Document

2. Project Plan Document

3. IEEE Guide for Software Quality Assurance Planning

4. IEEE Standard for Software Quality Assurance Planning

5. agentMom User's Manual

6. Scott A. DeLoach, Mark F. Wood & Clint H. Sparkman, Multi-agent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, Volume 11 no. 3, June 2001.

Management

Organization

Supervisory Committee:

Dr. Scott A. DeLoach

Dr. Daniel Andresen

Dr. Virg Wallentine

Major Professor:

Dr. Scott A. DeLoach

Developer:

Madhukar Kumar

Formal Technical Inspectors:

Cem Oguzhan

Esteban Guillen

Responsibilities

Supervisory Committee:

The committee would be responsible for attending reviews at the end of each phase and provide feedback and suggestions.

Major Professor:

In addition to the responsibilities of the supervisory committee, the advisor will be responsible for supervising and evaluation of work done by the developer on a weekly basis.

Developer:

The developer will be solely responsible for all the tasks outlined in the project plan and for the implementation of the final project. The developer will perform all tasks under the supervision of the Major Professor and will meet with him every week to show him the progress in work and also take further instructions and suggestions from him.

Formal Technical Inspectors:

The formal technical inspectors will submit a formal report after inspection on the architecture design artifacts.

Tasks

The tasks to be performed for the entire project and the breakdown of individual tasks have been defined in the project plan document and the Gannt Chart.

Documentation

The required documentation is outlined in the MSE portfolio handbook for the MSE students at the CIS department at Kansas State University. These documents would include a vision document, a project plan document, a software quality assurance plan document, a test report and a user manual.

These documents will be posted on the web (http://www.cis.ksu.edu/~madhukar/) and also be presented to the committee members during the three presentations and demonstrations. The committee members will evaluate the documents.

Standards, Practices, Conventions and Metrics

Documentation standard – Wherever applicable, IEEE standards and practices will be used to develop all documents for the project.

Coding standard – The environment (agentTool) will ensure that all standard coding practices will be used like indenting all control structures and embedding descriptive comments and invariants in all the methods and attributes of the separate classes.

Reviews and Audits

The reviews will be conducted by the committee members as outlined in the MSE portfolio handbook of the CIS department at Kansas State University. This includes three presentations and submission or required documentation to the committee members at the beginning of each presentation. The three presentations would also include demonstration of prototypes intended to establish the feasibility of the various aspects of the project.

Test and problem reporting

The test report documents will outline all the tests conducted and the results of the test. All problems will be reported to the committee chair, Dr Scott DeLoach. Significant problems will also be reported on the time logs posted at the project web site.

Tools, techniques and methodologies

The following tools will be used in the project

- Software Development Environment
  - agentTool
  - Java (java.net packages)
  - Visual Basic Application
  - MS Visio (for developing object model and other design artifacts)

Deliverables

The following deliverables will be submitted at the end of the project.

- agentTool maml file.
- Java source code with Javadoc API HTML documents.
- Source code for MS Word macros.
- Design artifacts – object model, use case documents.
- Vision document.
- Project Plan document.
- Software Quality Assurance (SQA) plan document.
- User Manual.
- Test report.

CHAPTER 6 – TEST PLAN

Test plan identifier

MSE-V1.0

Introduction

The purpose of this document is to outline the plan for testing the critical use cases and all functionalities of the Multi-Agent Research Tool (MART) software. The document will also describe the activities related to testing the software and the environment and tools that will be used to test the software.

The following documents will be used for reference:

Vision Document (sections 3.1 and 3.2 describing Software Requirements Specification).

Software Quality Assurance Plan (SQAP).

Test items

The following items of the MART software will be tested. The test will be performed after the second presentation and when the coding is complete.

The following item of the MART software will be tested:

The critical use cases

Features to be tested

The following features of the MART software will be tested. The description of each feature is described in the corresponding sections of the Vision document.

Type article in MS Word (Section 3.3.1 in Vision Document)

Search for research material (Section 3.3.2 of Vision Document)

Display research material (Section 3.2.3 of Vision Document)

Features not to be tested

Not applicable since all features will be tested.

Approach

The developer will use the specific requirements part of the vision document to test the above mentioned features of the MART software.

Type article in MS Word

The tester should be able to install the provided macro in MS Word and none of the preset preferences and page settings should change or stop working.

Search for research material

The user should be able to click on the button on the menu bar that says, "Search and Save" and that should automatically read all the relevant keywords already typed by the user in the article. Once the search is complete without any errors, a message should appear on the screen that the search was OK. When the search is complete a file called keywords.txt should be created in the My Documents folder of the computer. Opening the file in any text editor should show all the keywords that were created by the MART software.

Display research material

The user should be able to click on the button (macro) on the menu bar in MS Word and that should open up Internet Explorer with the displaying all relevant results. The user should be able to click on links that will open the relevant websites in the same window.

Functional Requirements Testing

All functional requirements will be tested against their requirements described under sections 3.2 of the Vision document.

Portability Testing

The developer team will try and run some of the agents from a Unix/Linux while running MS Word under Windows operating system. Any installation or running

problems that are encountered will be recorded in the Test log document.

Item pass/fail criteria

The MART software must be able to pass the tests for all the critical uses cases and the functional requirements described under section 3.2 of the Vision document. Each feature will be considered to be passed if it satisfies the corresponding requirement in the Vision document and failed if any or none of the behavioral expectations are met as described.

Suspension criteria and resumption requirements

Suspension criteria

If the user is unable to get any search results for an article after typing at least a paragraph then the testing will be suspended till the bug is traced and corrected. While testing new versions or releases of the MART software, testing will be suspended if any of the features of the previous release fails the test.

Resumption requirements

Testing will be resumed if and when the search function works adequately and when the user is able to view some relevant results. When testing for new releases, the testing will resume when all previously passed features of the previous release is considered to be passed.

Test deliverables

The following artifacts will be produced after the tests are conducted on the MART software:

Test Plan

Assessment report

Environmental needs

This section will outline the necessary and desired properties needed for the test

environment.

Physical Characteristics

System Software

All tests will be performed using MS Word running under Windows XP or Windows 2000. Some of the agents will be tested while running under Unix or Linux platforms.

CHAPTER 7 – ASSESSMENT EVALUATION

Test cases overview

Test Id	Test Unit	Test Case	Result/Comments
1. Specific requirements Typing in Word	User Interface. MS Word	Do all existing Word features work?	Passed
2. Specific requirement Search	The whole integrated system	Does MART create a results page if some words are typed in MS Word	Passed.
3. Specific requirement Display Result	The whole integrated system	Does the result page open up in Internet Explorer when user presses DisplayButton in Word?	Passed.
4. Portability testing	ProducerPackage, GooglePackage and TeomaPackage	Do the agents run on Windows, Unix and Linux	Yes, except on the Linux machines in the CIS lab where the machines are

		machines?	configured to drop multicasting packets. The agents work fine on Suse 8.1 and Windows XP professional on developer's laptop.
5. All three fields empty in User Preferences dialog menu	UserPref.java	No input for keywords, website and file format	Error message – "Not a valid input." Passed after modifications were made to UserPref.java
6. Invalid port number	AgentGoogle.java	Port number already taken by some other service	Error message. "Bad port number"
7. No keywords inputted by user.	Agent Manager package	No keywords inputted in MS Word and in Dialog box triggered by FilteredSearch	Error message- "Sorry, no keywords inputted."

		button	
8. Invalid port number	AgentTeoma.java	Port number already taken by some other service	Error message. Agent does not start
9. Invalid port number	AgentProducer.java	Port number already taken by some other service	Error message. Agent Producer does not start

Test id 1 - Typing in Word: After integrating the macro buttons for MART inside MS Word, the developer tested if typing in Word was still possible and if all the features still worked after MART was completely installed and running. The developer found that MS Word continued to work without any errors and/or change in existing configurations.

Test id 2 – Search: When the user types out some words inside a document in MS Word and/or in the keywords field after pressing the FilteredSearch Result, a file called final_result.html gets created at the specified folder. The developer found that some of the keywords that were inputted by the user are also found in the search results. This was one of the properties that needed to be true all the time and was validated using a USE model in phase 2 of the project.

Test id 3 – Display result: When the search has been conducted and there are no error messages and the user presses the DiplayResult button in MS Word, Internet Explorer browser opens up and the result file, final_result.html is displayed in it. If the user clicks on one of the URLs in the document, he/she is taken to that website inside the same

browser window.

Caveats for test id 2 and 3: For the tests id 2 and 3 to pass, the following conditions must be true-

1. User must have inputted keywords either by typing something in MS Word or through FilteredSearch user menu.

2. There must be at least one Sleuth agent (AgentGoogle or AgentTeoma) and one AgentProducer running on the same machine or the same network.

3. The machines hosting the Sleuth Agents must be connected to the Internet.

4. The firewall on the user's machine (that has MS Word running on it) should have the Windows and/or personal firewalls either switched off or configured to send/receive multicasting packets.

Test id 4 – Portability testing: The developer tried to run AgentGoogle, AgentTeoma and AgentProducer on the following machines – a Windows 2000 machine, Windows XP Professional machin, a Suse 8.2 Linux machine and a Solaris machine. All the times, the Agents started and worked properly and returned results. A caveat for the above test is that the machines running the Agents should have java 1.4 installed on them.

Test id 5, All three field empty in User Preferences dialog menu: When the user is in MS Word and presses the "FilteredSearch" button, he/she is presented with a user interface which asks for the user to input one or all three of the following fields: keywords, website and file format.

Initially when the developer conducted a test of not inputting any of the values in the field, the Manager Agent still started and tried to contact the Sleuth Agents with an empty string as keywords. This is not desirable so the developer made changes to the UserPref class and now if the user does not fill out all three of the fields then he/she will get an error message "Not a valid input" and the system will exit. The user has to

complete at least one field in the dialog box to start the AgentManager. If the user does not type any keywords, the system will extract keywords that have already been typed in the MS word article.

Test id 6, Invalid port number: When the user starts the AgentGoogle class, he/she is prompted for the number of instances of the Agent that the user wants to start followed by the port numbers that those Agents should "listen" on. These port numbers have to be free and not taken by another service already running on the server. For example port numbers 0 to 1024 are taken by known services and several port numbers from 1024 to 7000 have been registered. If the user tries to use one of the above mentioned port numbers, he/she will get a message " ** Bad port number given to Listener constructor for agent AgentGoogle" and the program will stop. To give a valid port number the user should try any number between 7000 and 65000.

Test id 7, No keywords inputted by user: When the user does not type anything in an open document in MS Word and/or also does not type any keywords from the FilteredSearch dialog, the AgentGoogle and AgentTeoma returned an NullPointerException.

This was corrected by adding code to the AgentManger. Now when AgentManager asks AgentKey to read the keywords, it checks whether it gets back a non null String (means the user did type some words). If the String returned by AgentKey is empty then the AgentManager outputs an error message - "Sorry, no keywords inputted" and then makes a graceful exit.

Test id 8, Invalid port number: When the user starts the AgentTeoma class, he/she is prompted for the number of instances of the Agent that the user wants to start followed by the port numbers that those Agents should "listen" on. These port numbers have to be free and not taken by another service already running on the server. For example port

numbers 0 to 1024 are taken by known services and several port numbers from 1024 to 7000 have been registered. If the user tries to use one of the above mentioned port numbers, he/she will get a message " ** Bad port number given to Listener constructor for agent AgentGoogle" and the program will stop. To give a valid port number the user should try any number between 7000 and 65000.

Test id 9, Invalid port number: When the user starts the Agent Producer, he/she is prompted for the port number that the agent should run on. If the port number is below 7000 then chances are that it already taken by some other service on that computer. If the user types a port number that is already taken by some other service, he/she will get a message " ** Bad port number given to Listener constructor for agent AgentGoogle" and the program will stop. To give a valid port number the user should try any number between 7000 and 65000.

CHAPTER 8 – USER'S MANUAL

System Overview

Multi Agent Research Tool (MART) is a multi-agent system that helps a researcher and/or author in conducting searches on the web while the user is typing the article in MS Word. The user, while typing the article can use the system by clicking one of the two buttons, **Search** and **FilteredSearch,** available in the MS Word menu. Each time the buttons are pressed, the system conducts a search, creates an HTML page and stores it on the user's computer. When the user is ready, he/she can press the **DisplayResult** button that will show the results page in Internet Explorer.

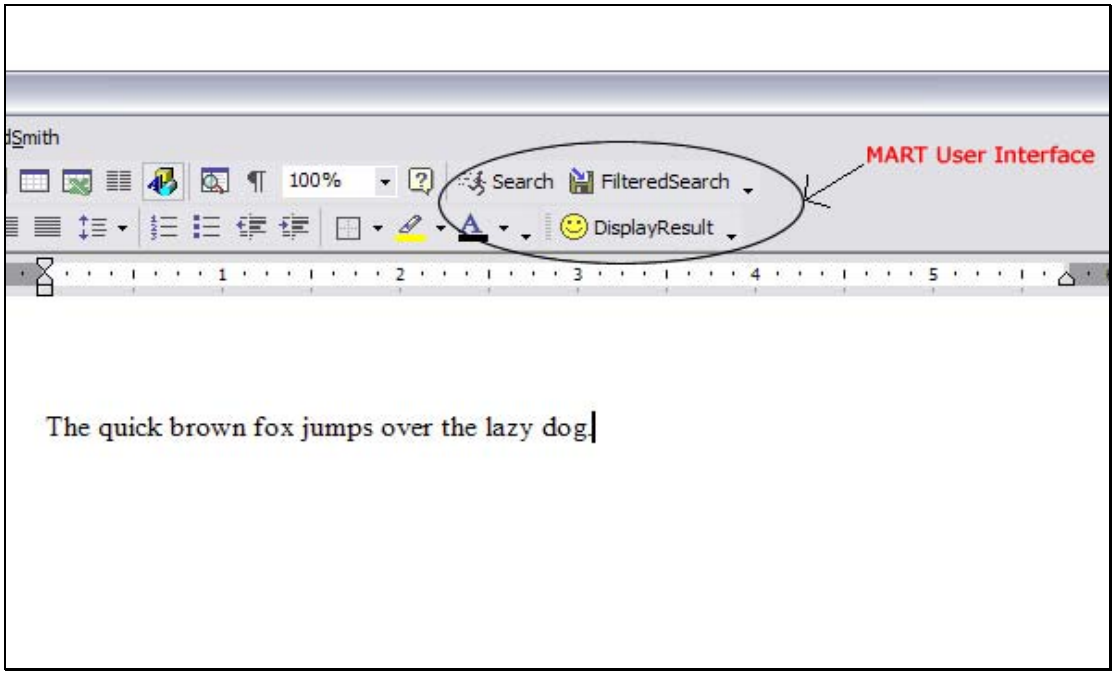


Figure 33 - User Interface

Using the System

To use the system, first run AgentGoogle and/or AgentTeoma and AgentProducer as detailed in the following steps and then start typing the article in MS Word. When you

are ready, just click on the **StartSearch** or **FilteredSearch** buttons. When you are finally ready to view the result, click on the **DisplayResult** button from the MS Word menu.

Running AgentGoogle

Descend into the directory where GooglePackage directory is installed and double click on AgentGoogle.bat file if you are in windows. If you are on a Linux machine, simply run the AgentGoogle.sh file.

You will now be prompted for the number of instances of AgentGoogle you want to run. When you type in the number of instances of AgentGoogle you want to run, you will be prompted for port numbers that the Agents should run on. Type the port numbers that are **not in use** by any other services on the computer.

Running AgentTeoma

Descend into the directory where TeomaPackage directory is installed and double click on AgentTeoma.bat file if you are in windows. If you are on a Linux machine, simply run the AgentTeoma.sh file.

You will be prompted for the number of instances of AgentTeoma you want to run.

When you type in the number of instances of AgentTeoma you want to run, you will be prompted for port numbers that the Agents should run on. Type the port numbers that are **not in use** by any other services and/or agents on the computer.

Running AgentProducer

Descend into the directory where ProducerPackage directory is installed and double click on the AgentProducer.bat file if you are in Windows. If you are on a Linux machine, simply run the AgentProducer.sh file.

Conducting Searches

Click on Search macro button from the MS Word and continue typing your article.

You will see a new DOS prompt window open up and some messages. When the final result page has been created, the DOS prompt window will go away on its own. You can continue typing in MS Word by minimizing the DOS prompt window.

The following figure shows how a user has typed a line in MS Word and then clicked on the Search macro button.

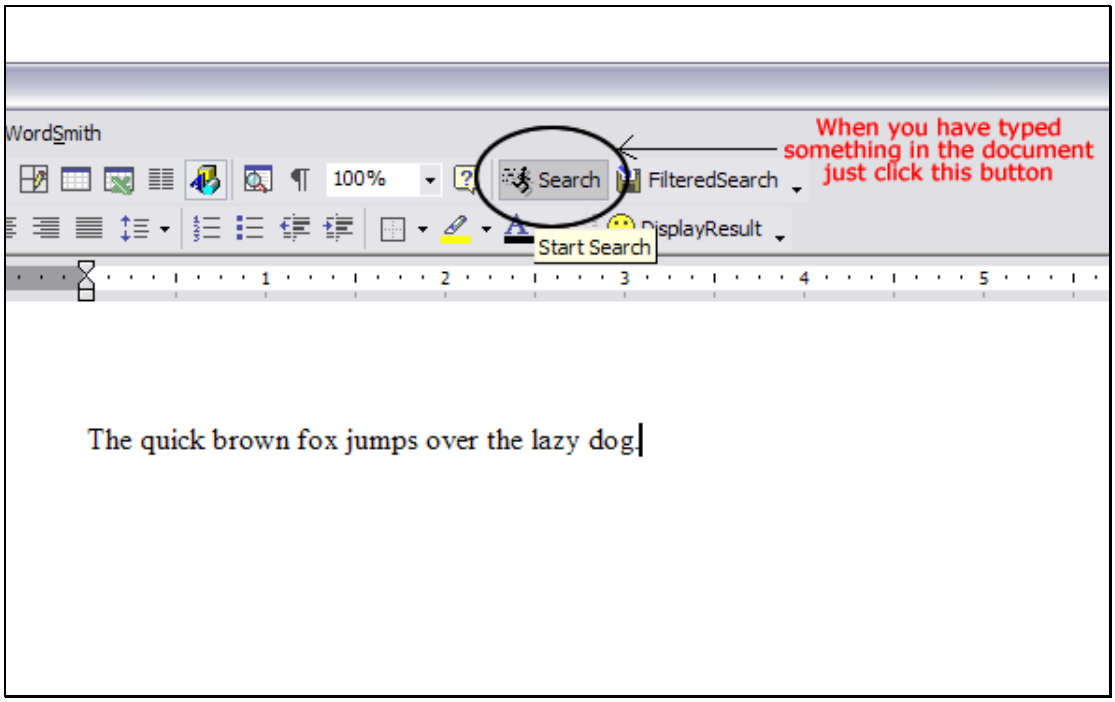


Figure 34 - Conducting search without preference

Conducting Filtered Search

MART can also be used to conduct searches with user defined preferences. To do this, simply click on the **FilteredSearch** macro button in the MS Word. You will see a dialog menu as shown in Figure 3. If you type some keywords in the first text field and

then give a name of the website and also specify the document format by typing d or p then MART will conduct a search only on those keywords on the website you specified and only in pdf or doc documents. For example, if you typed everything exactly the same preferences as shown in Figure 4 then MART will conduct a search only on words software agents and restrict the search to the citeseer.nj.nec.com website and search only pdf documents.

If you leave any field empty in the dialog menu, for example if you don't type in the keywords then MART will automatically pick up keywords from whatever you have typed out in MS Word. However, if you leave all three fields empty and then press the OK button then you will get an error message.

When you are typing out the website address, please ensure that you do not type http:// prefix in the website text field.

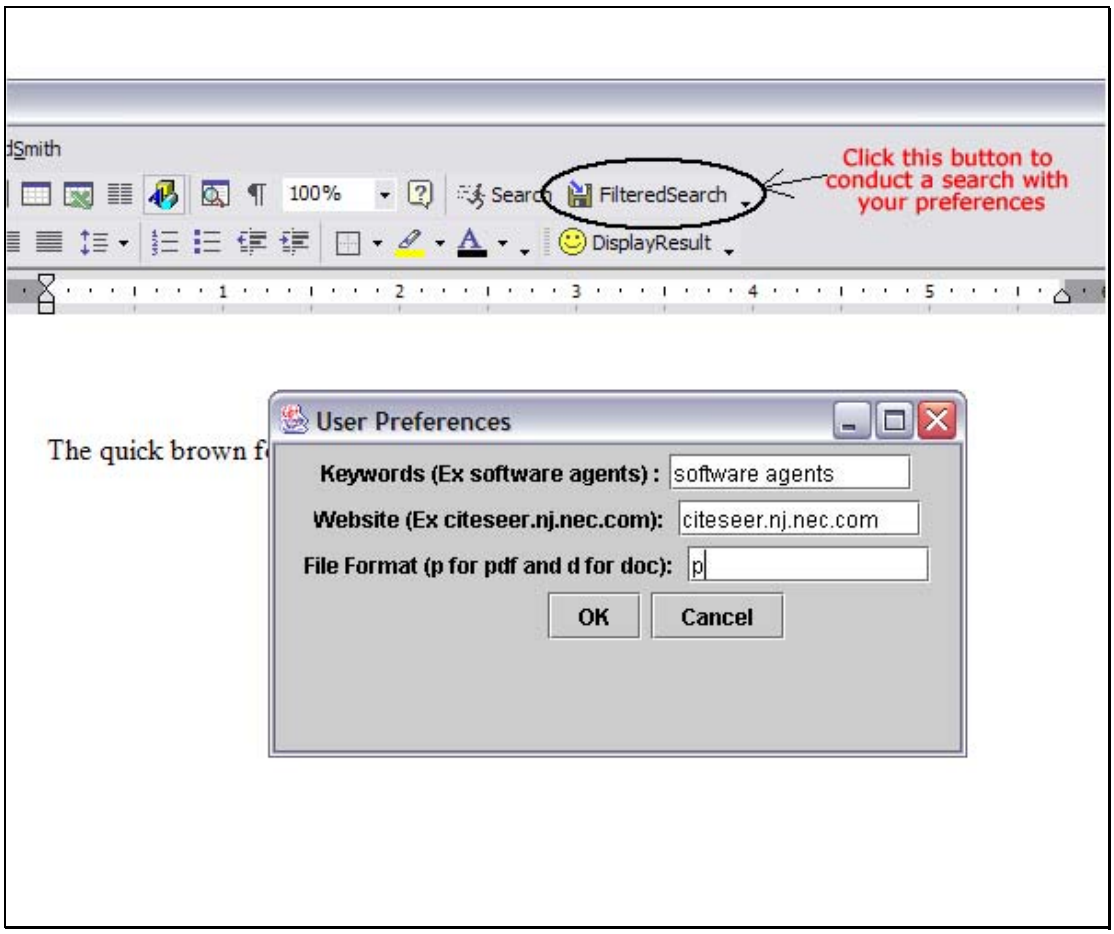


Figure 35 - Conducting search with preferences

Viewing search results

When you are ready to view the results, simply click on the DisplayResult macro button and Internet Explorer will open up with the final results file displayed in it. You can click on any one of the links to navigate to the relevant websites.

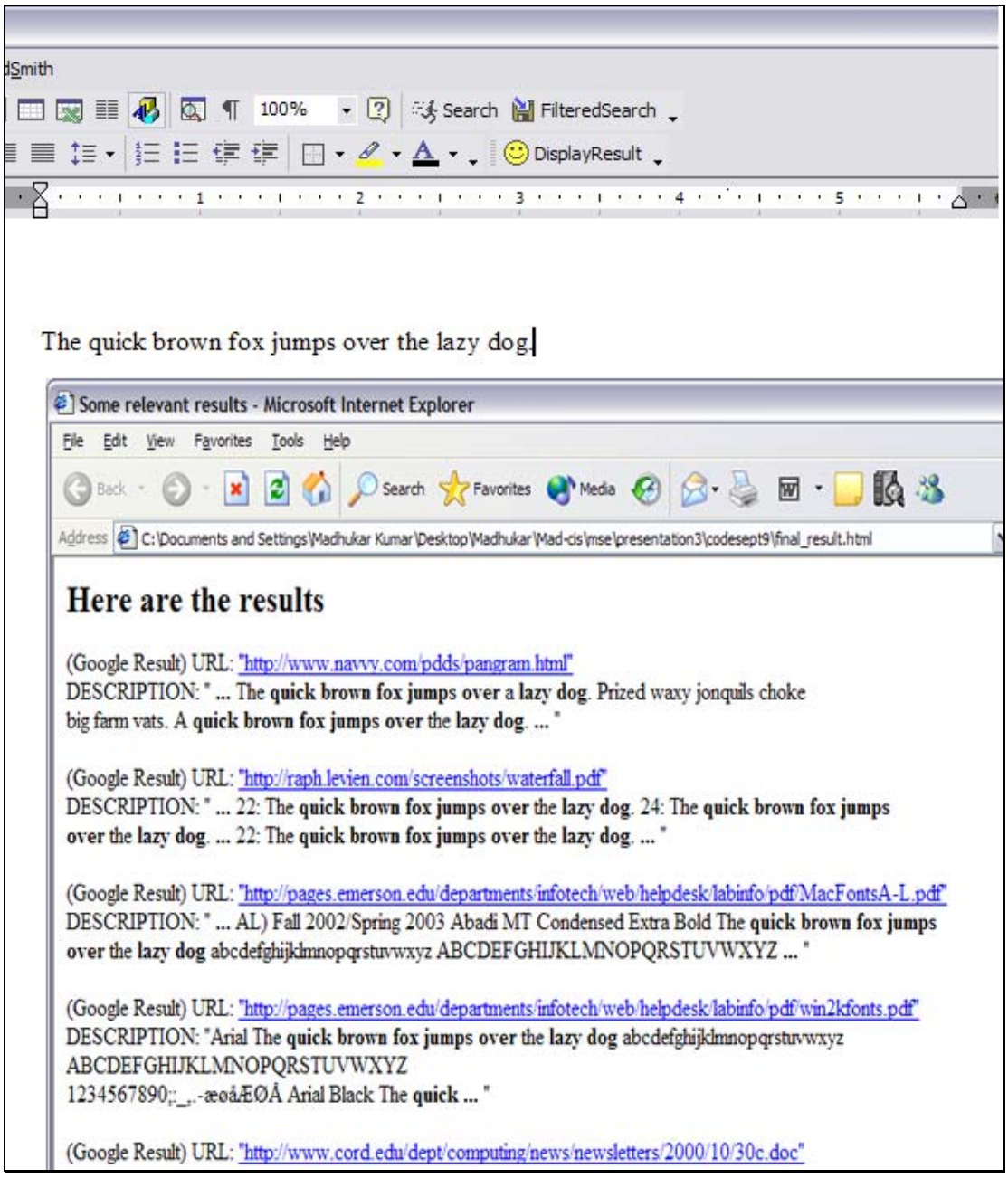

Smith
100%
Search
FilteredSearch
DisplayResult
The quick brown fox jumps over the lazy dog.
Some relevant results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
C:\Documents and Settings\Madhukar Kumar\Desktop\Madhukar\Mad-cis\mse\presentation3\codesept9\final_result.html
Here are the results
(Google Result) URL: "http://www.navvy.com/pdds/pangram.html"
DESCRIPTION: " ... The quick brown fox jumps over a lazy dog. Prized waxy jonquils choke big farm vats. A quick brown fox jumps over the lazy dog. ... "
(Google Result) URL: "http://raph.levien.com/screenshots/waterfall.pdf"
DESCRIPTION: " ... 22: The quick brown fox jumps over the lazy dog. 24: The quick brown fox jumps over the lazy dog. ... 22: The quick brown fox jumps over the lazy dog. ... "
(Google Result) URL: "http://pages.emerson.edu/departments/infotech/web/helpdesk/labinfo/pdf/MacFontsA-L.pdf"
DESCRIPTION: " ... AL) Fall 2002/Spring 2003 Abadi MT Condensed Extra Bold The quick brown fox jumps over the lazy dog abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ ... "
(Google Result) URL: "http://pages.emerson.edu/departments/infotech/web/helpdesk/labinfo/pdf/win2kfonts.pdf"
DESCRIPTION: "Arial The quick brown fox jumps over the lazy dog abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890;:_,.-æøåÆØÅ Arial Black The quick ... "
(Google Result) URL: "http://www.cord.edu/dept/computing/news/newsletters/2000/10/30c.doc"

Figure 36 - Results page after clicking DisplayResult button

System Configuration

Requirements

The following are required to use MART

1. A Windows machine with MS word installed on it. This will be referred to as the host machine.
2. JDK 1.4 (java) installed on the host machine and the other machines that will be used to run agents.
3. The ManagerPackage and KeyPackage should be installed and run from the host machine.
4. All agents of MART are capable of running from either the host machine or from any other machine(s) (with jdk1.4 installed on them) located on the same network.

Installing Macros in MS Word

To install the Macros in MS Word, the user should perform the following steps:

1. Unzip mart.zip in your system root folder at c:\
2. Open MS Word.
3. Navigate to mart folder and open mart.dot
4. Click on Tool->Macro->Macros.

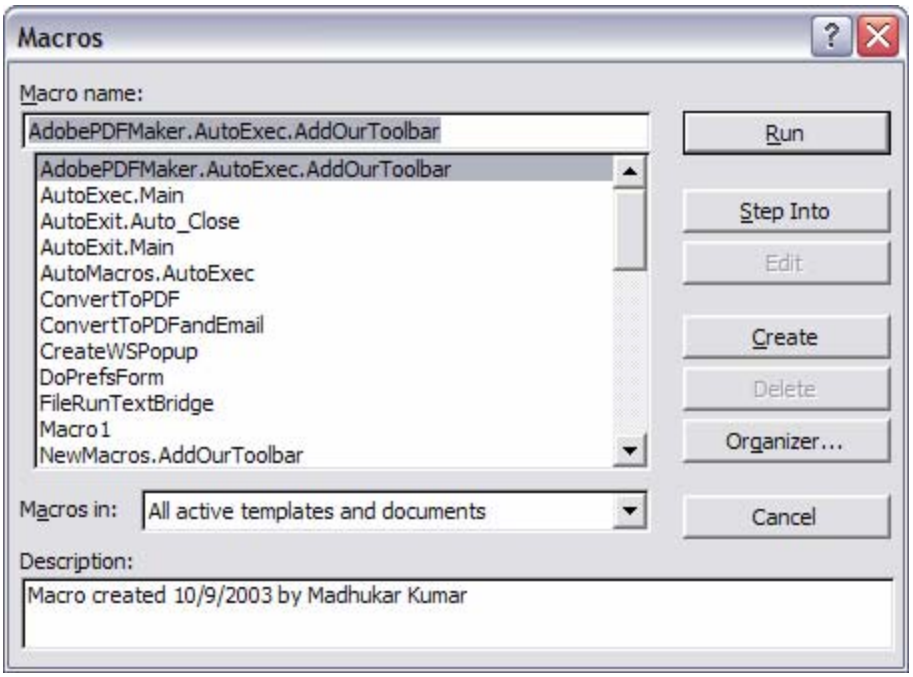


Figure 37 – Installing Macros-1

5. Click on the **Organizer** button

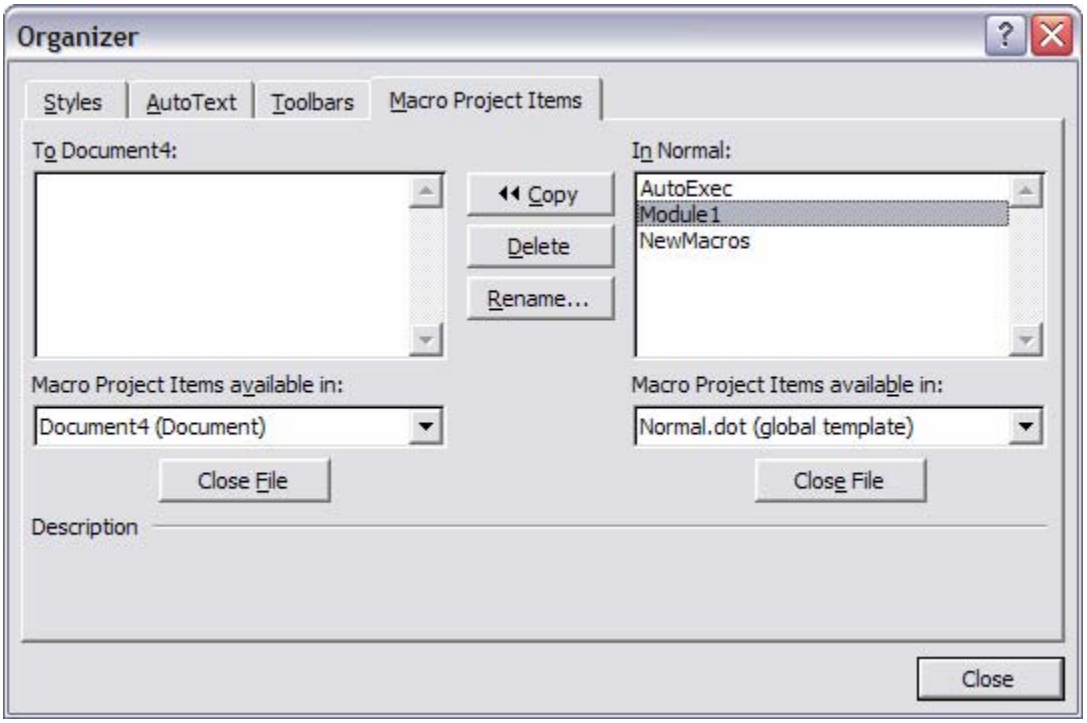


Figure 38 – Installing Macros-2

6. Now click on Module1 and click on the **<<Copy** button so that Module1 now also shows in the left panel as in the following figure

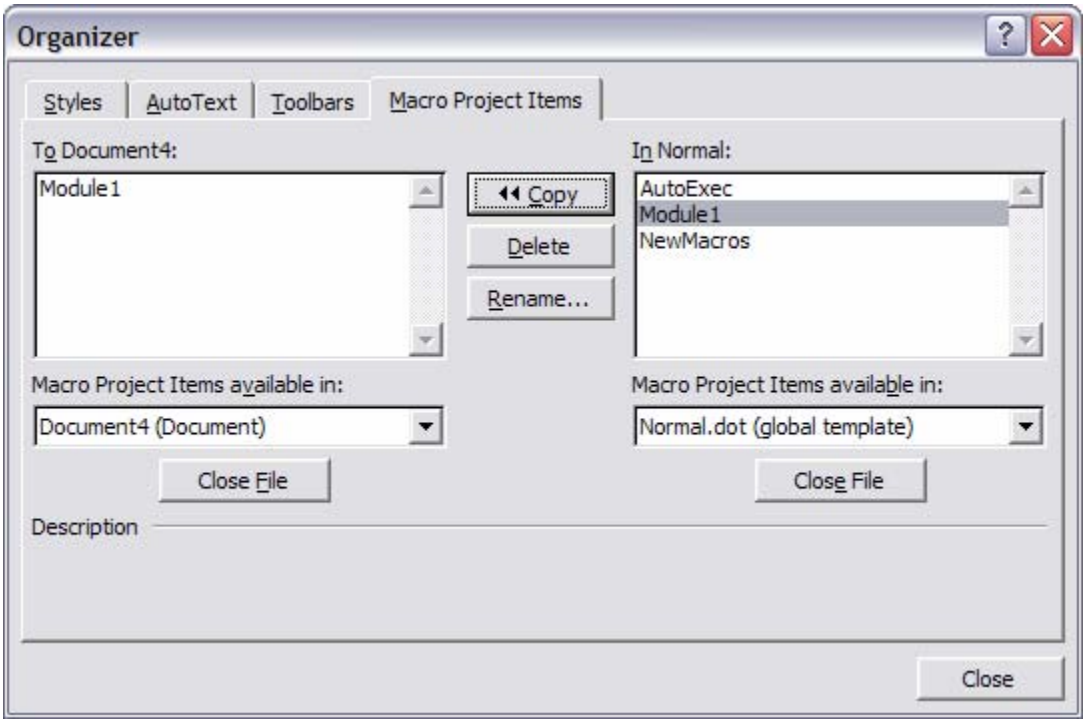


Figure 39 – Installing Macros - 3

7. Now right click any where near the menu in MS Word and then click **Customize**.

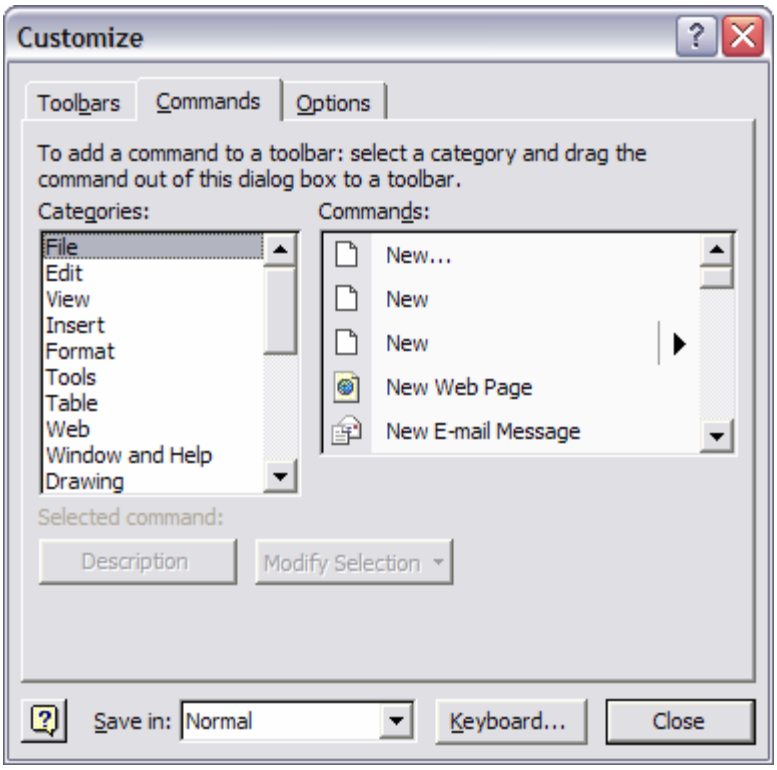


Figure 40 – Installing Macros - 4

8. Click on **Macro** button on the left panel

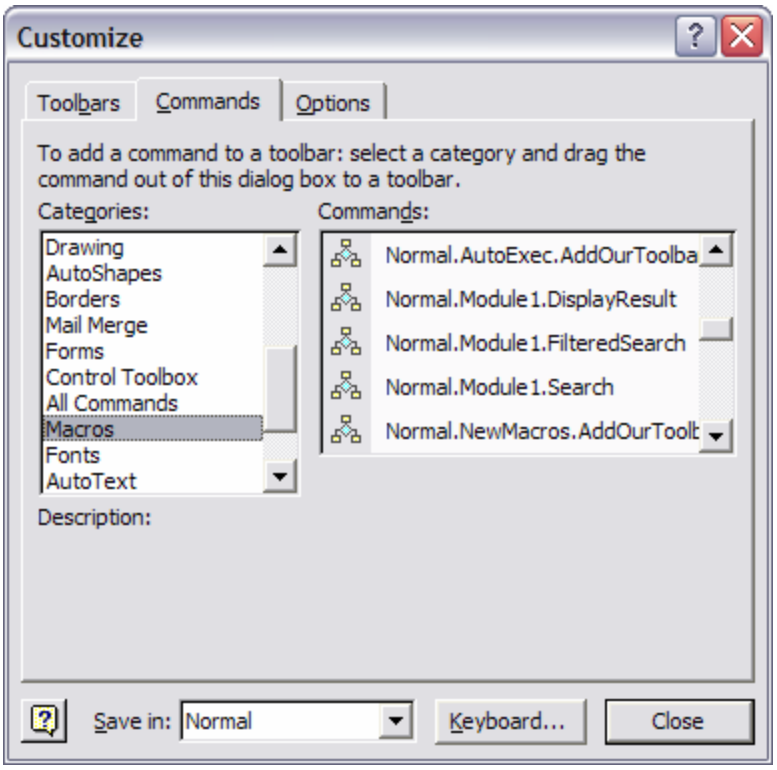


Figure 41 – Installing Macros - 5

9. Scroll down in the right panel till you see Module1 macros as shown in the above figure.
10. Now drag Normal.Module1.Search and drop it anywhere in your MS Word menu.
11. Right click on it and rename it to Search and choose an icon image of your choice.
12. Do the same steps for **Normal.Module1.FilteredSearch** and **Normal.Module1.DisplayResult** and rename them **FilteredSearch** and **DisplayResult** respectively. MART in ready to be used from within MS Word.

Installing Agents

Unzip mart.zip and copy the GooglePackage directory and TeomaPackage directory to any machine on your local network. The GooglePackage consists of a file called googleapi.jar. Make sure this is in the classpath of the system where you plan to run this agent or unjar it in the directory where you see it.

For installing AgentProducer, simply copy the file to any machine on your local machine where you plan to run it from.

Make sure that ManagerPackage and KeyPackage are located on the same machine that you will be using to run MS Word.

CHAPTER 9 – PROJECT EVALUATION

Problems encountered

Multicasting

During the end of the second phase of the project a new requirement was added. The requirement was that the AgentManager should somehow be able to "know" of all the Sleuth agents that are running on the local network. To implement this, the concept of multicasting was explored and finally implemented in the project. The main problem that the developer faced while implementing multicasting in the project was that while it worked in some scenarios, it stopped working in other scenarios without throwing any errors.

Three lessons were learnt from this problem. First, multicasting can work only on a single network. To make multicasting work across networks, the developer needs to have access to the routers so that they can be enabled to carry forward the multicasting packets.

Second, to make multicasting work on a Windows machine, the computer has to be connected to the Internet. Multicasting will not work on a Windows machine if a cable (or wireless card) is disconnected. The Windows' native Firewall also has to be either switched off or configured (to allow packets transmission of certain port numbers) to make multicasting work on the machine.

Finally, if the developer is trying to send and receive multicasting packets on a Linux machine, he/she has to ensure the machines are configured not to drop multicasting packets.

Debugging

As the system grew in size, it became incredibly hard to trace bugs since MART is a multi-agent system and the agents were behaving as independent systems running on

different machines.

The lesson learnt from this problem was that with each increment the whole system should be compiled and tested before moving on to the next increment. This is time consuming but useful in debugging parallel programs interacting with each other.

Numbers of lines of code

The original estimate of the number of lines of code for MART was based on the developer's earlier experience in developing a multi-agent system using AgentTool. The original estimate was 1500 lines of code during the first phase of development. During the second phase, the developer estimated that the actual lines of code could be near 2550. However, the final size of the entire package turned out to be almost 3,000 lines of code. The aberration in the original estimate and the actual lines of code in the final product was because the original estimate of 1500 lines of code was wrong due to developer's inexperience in estimating size during the first phase of the project. During the second phase the developer estimated the final software to be around 2550 and this would have been fairly accurate but for the fact that a new requirement was added during second presentation (multicasting). Due to addition of some new classes which had not been taken into account in the second phase, the final product had about 350 more lines of code than the estimated 2550 in the second phase.

The lines of code according to packages are listed below.

TeomaPackage = **469**

GooglePackage = **545**

ProducerPackage = **576**

ManagerPackage = **859**

KeyObjectPackage = **64**

AgentKeyPackage = **450**

Total = **2963 lines of code**

Cost Estimate

Though the final lines of code of 3000 was almost double the original estimate of 1500 lines of code, the developer actually managed to finish a few days ahead of the schedule. This is primarily because at the end of second phase, the project plan and the cost estimate were refined. Using COCOMO, the developer estimated that the actual lines of code would be around 2550 and not 1500 and hence the effort would actually be 8.4 months instead of 4.9 months (estimated during the first phase). This also meant that the time of development would now be 7.9 months instead of 4.6 months as derived in the first phase of development. After the new effort and time of development was derived in the second phase, the developer set the new final deadline to be on November 27, 2003 as against Aug 2003 estimated during the first phase. Since the implementation and the first phase of documentation finished by October 7, 2003, the project actually finished ahead of schedule and the cost estimation in the second phase turned out to be fairly accurate.

Project Duration

The following table details the expected time of finish for each phase and the actual finish time.

	Expected Finish Time	**Actual Finish Time**
Phase 1	Feb 5, 2003	Feb 4, 2003
Phase 2	August 20, 2003	August 28, 2003
Phase 3	Nov 24, 2003	Oct 31, 2003

The following table details the break down of activities in each development phase

	Phase 1(Spring 2003)	**Phase 2 (Summer 2003)**	**Phase 3 (Fall 2003)**
Meetings	130 min	430 min	150 min
Documentation	589 min	626 min	771 min
Coding	230 min	189 min	1239 min
Research	80 min	0 min	90 min
Keeping Timelogs	74 min	30 min	95 min
Designing	0 min	523 min	0 min
Integration, debugging, troubleshooting	30 min	0 min	80 min

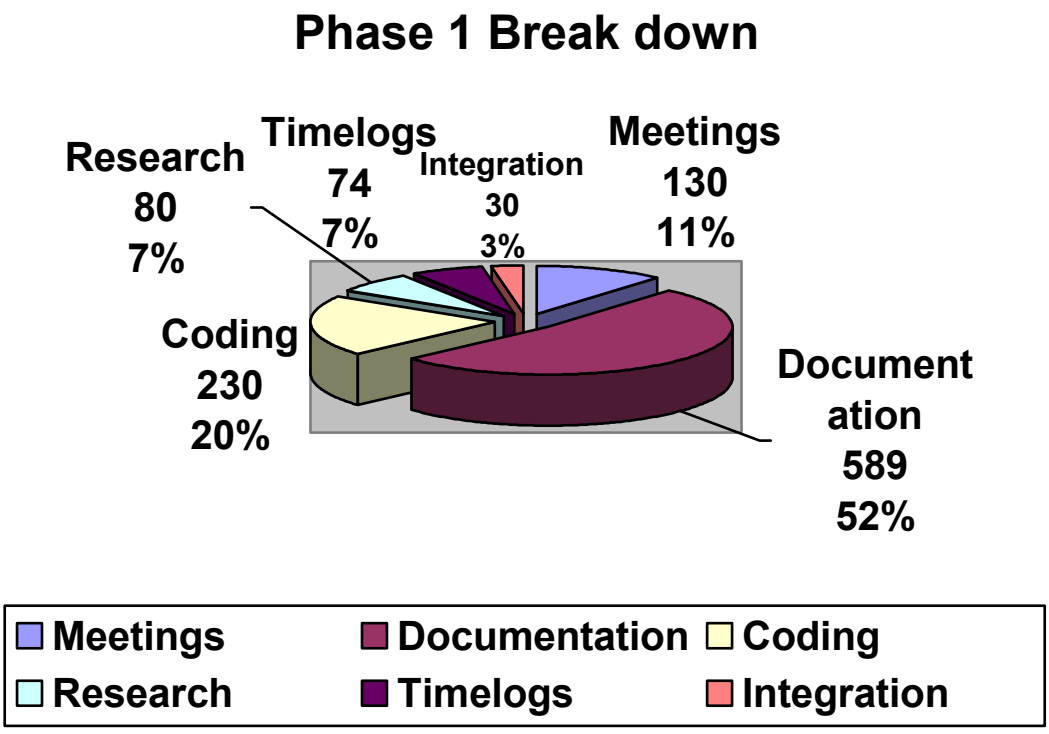


Figure 42 - Phase 1 Break down

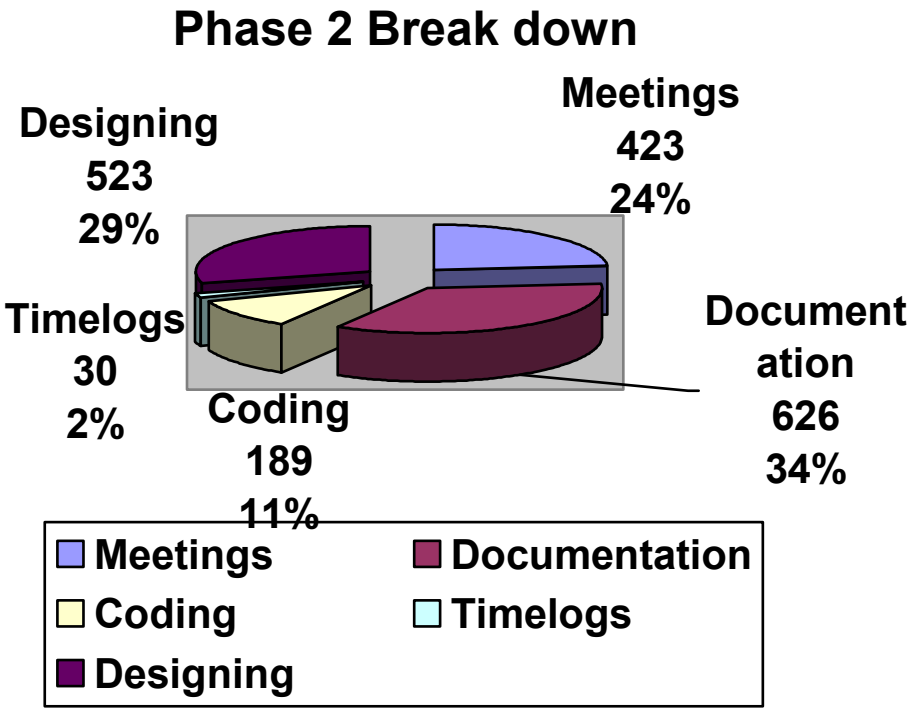


Figure 43 - Phase 1 Break down

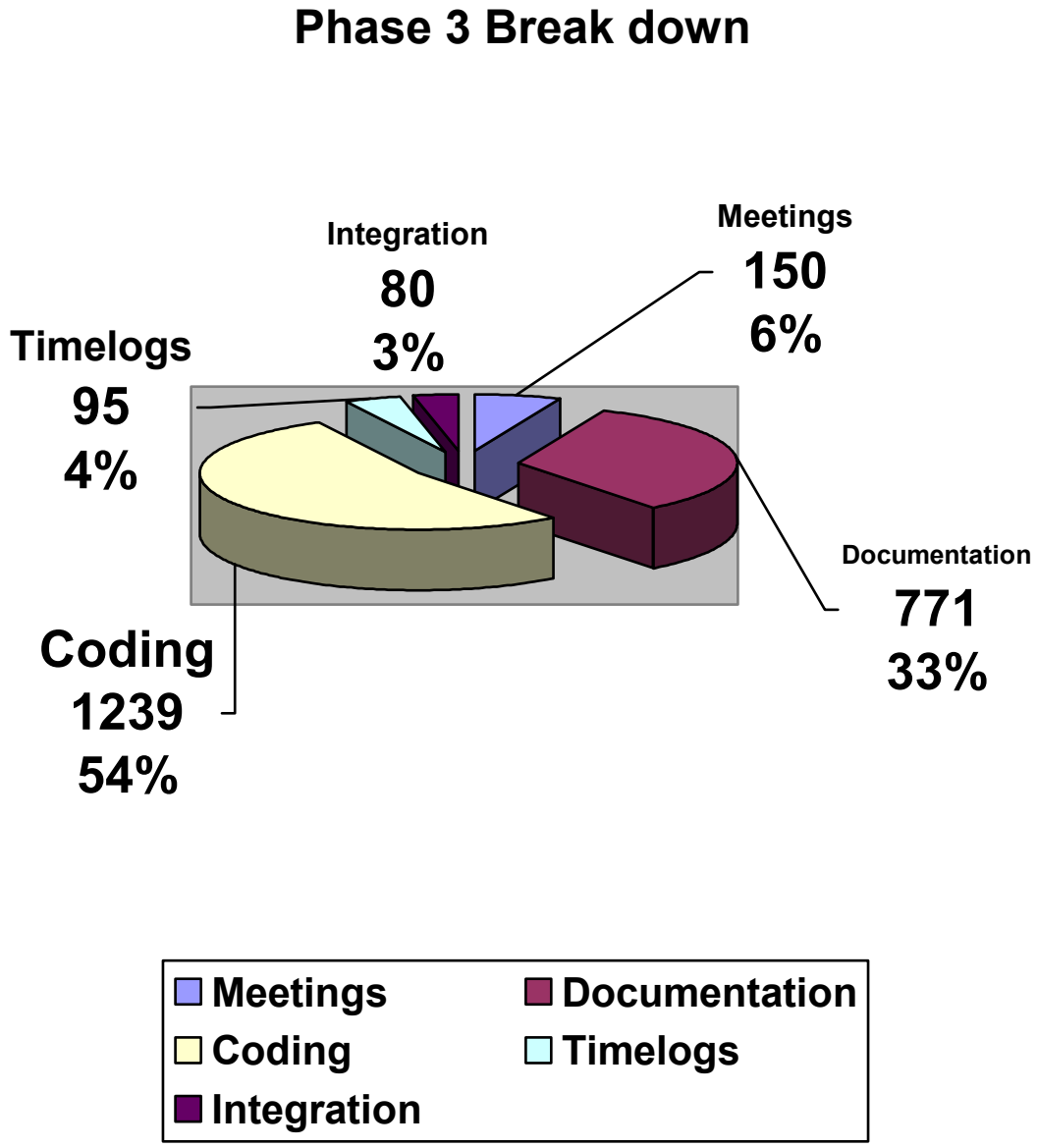


Figure 44 - Phase 3 Break down

Lessons Learnt

Usefulness of methodology

One of the biggest advantages of using Agent Oriented Software Engineering (AOSE) to develop the system was the fact that agents are fairly independent programs and when one goes down the entire system is still functioning.

Since the developer had used agentTool to develop a multi-agent system before, he was aware of the smaller nuances of using the tool and had no big learning curve. This greatly cut down the implementation time.

It would be nice if agentTool allowed the design of some external components into the system during the design and analysis phase and it would be also nice if it had multicasting feature built into it. The developer is aware that both these features are being built into agentTool next version and eagerly awaits its release.

Usefulness of reviews

The reviews helped the project move in a tightly regimented process which is similar to the process of building software in a commercial setting. The reviews also helped the developer in getting valuable feedback from committee members on how to smooth the edges of the final product and ensure that the product is actually usable to the end user.

Product quality

The developer believes that given the project and deadline constraints, the final product is fairly efficient and bug free. However, the product could still be improved on two counts. First, the Sleuth agents could be made faster by cutting down communication overheads and also by exploiting result caching on the user's machine. Second, the final result page could be refined further so that the search results are more relevant to what the user types in the MS Word document. The ranking of keywords could also be refined further to make the process of making keywords more efficient.

REFERENCES

IEEE Std 829-1983, "IEEE Standard for Software Test Documentation". 1983 Edition, IEEE, 1983.

IEEE Std 828-1998, "IEEE Standard for Software Configuration Management Plans". 1998 Edition, IEEE, 1998.

IEEE Std 730-1998, "IEEE Standard for Software Quality Assurance Plans". 1998 Edition, IEEE, 1998.

IEEE Std 730.1-1995, "IEEE Guide for Software Quality Assurance Planning". 1995 Edition, IEEE, 1995.

_____, "Java 1.4 API".

Pressman, Roger S. "Software Engineering: A Practitioner's Approach". Fifth Edition, Mc Graw Hill, NY, June, 2001.

Scott A DeLoach. "agentMom User's Manual." July 2000.

Scott A. DeLoach, Mark F. Wood & Clint H. Sparkman, Multi-agent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, Volume 11 no. 3, June 2